CONTINUOUS SPEECH PHONEME RECOGNITON USING NEURAL NETWORKS AND GRAMMAR CORRECTION

By

Wai-Tat FU

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Electronic Engineering

The Chinese University of Hong Kong

JUNE 1995

Abstract

Utilization of context information of speech is essential in continuous speech recognition, even just to attain reasonable results. This thesis addresses the use of context information in continuous speech at both low and high levels. At the low level (signal level), several current approaches are reviewed and recurrent neural network is found to have good ability in handling context information across speech frames efficiently. At the high level (symbol level), finite state grammar is used and shown to be good at capturing the structure of phoneme sequences. An integrated system consisting of neural network and finite state grammar with some postprocessing of the neural network pattern classifier shows comparable results to the existing systems. Although the dataset (subset of the TIMIT) used may not be large enough to provide a good coverage of actual English utterances, it is believed that with sufficient computational resources, the same or better result could be obtained.

Contents

Chapter 1

INTRODUCTION

Automatic recognition of speech by machine has been a goal of research for more than four decades. In the past twenty years, many speech recognition strategies have been proposed and implemented. However, we are still far from achieving the desired goal of a machine that can understand spoken discourse on any subject by all speakers. The basic problems of speech recognition are described next.

1.1 Problem of Speech Recognition

1. *Lack of knowledge in acoustics, phonetics and modelings of speech*

   The speech signal is a slowly time varying signal in the sense that, when examined over a sufficiently short period of time, its characteristics are fairly stationary; however, over long periods of time the signal characteristics change to reflect the different speech sounds being spoken. It is arguable that whether the models in currect speech recognition systems

are truly representing the quasi-periodic nature of speech. Besides, the variations of speech could be very large. Even for the same person, very different speech waveform may be produced for the same word in different situations, let alone different speakers. While the signal conveys linguistic information, this information is often encoded in such a complex manner that the signal exhibits a great deal of variability. The knowledge of the relationship between the speech signal and the complex underlying linguistic forms is still very limited. When a phonetician is asked to analyse an utterance, the symbol string he produces is not a time-centred representation. The symbols will stand for widely differing time periods, and it will be more or less arbitrary at what instant one symbol stops and the next starts. One simply cannot answer the question "what was being said at instant t?", not because of the deficiency of the analysis technique, but is a reflection of the nature of speech [9]. The symbol sequence is an abstraction of an explanation of the acoustic evidence, each symbol is an object which stands for a structure of organised behaviour. The phoneme string is the top level of an object centred representation, which makes elements of the explanation and qualitative time relationship between these elements explicit.

2. *Continuous vs isolated and connected speech*

In isolated word recognition, word boundaries are known, and can be used to determine their similarity (or the distance between patterns) directly. For connected speech recognition, words or digits are uttered connectedly. This implies that the word boundaries are clear but not known as in the

case of isolated word speech recognition, and thus creates an additional problem [31, 25]. Continuous speech recognition is significantly more difficult than isolated and connected word recognition. Its complexity is a result of the properties of continuous speech. First, word boundaries are not known and even unclear and therefore difficult to find [10]. Also, the co-articulatory effects, which are the alterations of acoustic properties of speech of a particular symbol due to its previous and following symbols, are much stronger in continuous speech, as our articulators cannot move instantaneously to produce unaffected phone[1] sequences. Worse still, content words (nouns, verbs, etc) are often emphasized, while function words (articles, preposition, etc) are poorly articulated. In particular, phones in function words are often shortened, skipped or distorted [22].

3. *Vocabulary size*

   The vocabulary size varies inversely with the system accuracy and efficiency; as more words will introduce more confusion as well as require more time to process [21]. In addition, for small vocabulary size speech recognition, whole word model can be used which has less acoustic variations and the variability due to different context is less severe [14, 26]. In order to obtain reliable whole word models, the number of word utterances in the training set needs to be sufficiently large, i.e. each word in the vocabulary should appear in each possible phonetic context several times in the training set, thus increasing the computational resources required when the vocabulary size is increased.

[1] A *phone* is usually a description of the acoustic properties of the symbol, while a *phoneme* is used in the linguistic sense.

4. *Speaker Dependence*

   A speaker dependent system uses speech from the target speaker to learn its model parameters and requires an inconvenient period for adaptation to each new speaker. A speaker independent system is trained once and for all, and must model a variety of speakers' voices. Speaker independence has been viewed as one of the most difficult constraints to overcome because most parametric representations of speech are highly speaker dependent, and a set of reference patterns suitable for one speaker may perform poorly for another speaker. In [22], it has been postulated that for the same task, speaker-independent systems will have three to five times the error rate of speaker-dependent ones.

5. *Understanding of speech utterance*

   For application purpose, what actually essential to speech recognition is whether an information is successfully transmitted. Even if the speech waveform is sucessfully transformed to a sequence of words, the information from the speaker still may not be properly conveyed. Usually, several constraints are imposed to guide the search for a "meaningful" sentence. Non-sensible sentences need to be filtered out from the search, but this creates a problem of how to judge whether the searched sentence make "sense" at all. In the lowest level, constraints are imposed by our vocal tract and the system of pronunciation of the language [5, 12]. In the middel level (syntactic level), words has to comply with the "rules" (grammar) of the language, and the underlying rules has to be already known to both parties during the message transmission. In the high level

(semantic level), meanings have to be conveyed through words. Even if the sentence obeys the syntactic constraints (the combinations of noun, verb etc are acceptable), it is still possible that meanings cannot be conveyed, and some understanding constraints have to be imposed.

1.2 Why continuous speech recognition?

The final goal of a speech recognition system is to be able to "understand" human discourses. Isolated and connected word recognition are just a first step towards a real speech recognition system. Of course, if the task is confined, like recognising a sequence of number or a single word instruction, isolated or connected speech recognition could be good enough. However, in some tasks, isolated or connected speech recognition is not sufficient. For example, if we want to build a machine which understands human speech for the handicapped who is unable to type, it is desirable to have a machine capable to interpret continuous speech. On the whole, a good speech recognition system should provide a benefit to the user to increase productivity, better human-machine interface and a more natural mode of communication. Since the usual communication between human beings are continuous speech, it is natural to aim at building a machine which can understand continuous speech.

1.3 Current status of continuous speech recognition

For continuous speech recognition, there are two ways to evaluate the whole systems, either in the "sub-word (phoneme)" level or in the "word" level. Since continuous speech recognition systems usually uses sub-word units (reasons are to be explained later), a direct way of evaluation is to measure the accuracies of classification of these sub-word units. But a good classification of sub-word units is only a first step, the final practical continuous speech recognizer should be able to classify words accurately. In this thesis, only *phoneme recognition* is concentrated on, and the word accuracy is only used as a reference.

Currently, the most successful continuous speech recognition systems are either adopting the statistical approach [17, 23] using hidden Markov model or the connectionist approach [2, 29] using neural network, details of which are to be discussed in chapter 2. In general, they deal with the problems mentioned in section 1.1 as:

1. Acoustics, phonetics and modelings of speech

   As mentioned before, the lack of knowledge in both acoustics and phonetics of speech is causing degradations to most speech recognition systems. In [40], it was argued that powerful speech-recognition systems in the last decade have attained high performance because they have effective mechanisms to model our ignorance. For example, in his view, dynamic time warping have an ignorance model for dealing with the effect of speaking rate on the speech signal. Since there are many inherent difficulties in

building a practical speech recognition system, a lot of assumptions have to be made in order to meet the constraints. For example, making a very accurate modeling to speech (if it exist) should be essential to the performance of the whole system, but the computation involved would be very heavy. As a compromise, an modeling is considered useful only when it satisfies a reasonable memory, computation and time usage. Therefore usually an apparently less sophisticated model is adopted in most speech recognition systems.

2. Continuous speech and vocabulary size

   As mentioned before, the disadvantage of using whole word models in continuous speech recognition is that the number of word utterances in the training set needs to be sufficiently large. In addition, with a large vocabulary the phonetic content of the individual words will inevitably overlap. Thus storing and comparing whole word patterns would be redundant because the constituent sounds of individual words are treated independently, regardless of their identifiable similarities. As a result, for large vocabulary speech recognition, sub-word units are usually chosen as a representation of speech, disregarding its high variability due to context [22, 32, 39, 21]. And it seems to be unavoidable that the increase in vocabulary size will lead to the increase in recognition error rate. For example, in [30], a two fold increase in error rate is reported when word templates were replaced with demisyllable units.

   There are several possible choices for sub-word units that can be used to model speech. These include the *phonelike units (PLU)* , which is the

basic phoneme set of English and there are about 60 PLUs for English. The *syllable-like units*, which basically uses vowel nucleus plus the optional initial and final consonants, and has about 10000 types for English. The acoustic units, which use some kind of codebooks to capture clusters of speech segments, and it was shown in [20] that a set of about 256 - 512 acoustic units is appropriate for modeling speech vocabularies. The problem of PLUs is that it is very context sensitive, even thought only 60 of them is enough to represent almost all vocabularies. The syllable-like units are the longest units and are the least context sensitive, but the problem is there are so many of them that makes training almost impossible. (We can see a relationship here between the number of units used and the degree of sensitivity to context). On the other hand, the linguistic interpretation of the acoustic units are almost nonexistent, which makes the knowledge from phonological and lexical level almost inapplicable. In practice, PLU units are usually chosen as it is the only possible units under the constraints of dataset size, feasibility of training and have existing knowledge from the studying of linguistics, in spite of the fact that they are context sensitive. As a result, it is necessary to apply measures to account for the context sensitivity of the PLU units in the recognizer.

3. Speaker Dependence

   In order to implement a speaker-independent word recognizer, the variations among speakers in pronouncing the same word must be accounted for. Attempts had been made to both finding a speaker-independent features in speech and forming statistics on the variability of the features both

in time and across talkers. So far only the speaker-independent isolated word recognition's performance was improved in this way. For example, in [33], a 48% accuracy for using one speaker's templates to recognize another speaker's speech is achieved, while a 90% accuracy is possible for speaker-dependent recognition. Owing to the increased variability, speaker independent systems are typically less accurate than speaker dependent systems [38]. Both systems have its applications in practice. Both types of systems have been built and studied extensively.

4. *Understanding of speech utterance*

For most command-and-control task, the understanding phase is not necessary as the vocabulary words are itself control signals. For larger task, the sequences of words obtained are required to be processed by a "language" model in order to comprehend the message encoded in the acoustic waveform. Usually building a language model needs a lot of a priori linguistic knowledge. As mentioned before, the use of language model can be divided into three levels, and current continuous speech recognition system uses the middle (syntactic) level most . The highest (semantic) level is usually task specific and not too much comparison can be made by that. Not too much effort has been put to investigate the use of linguistic information to the lowest level (phoneme symbol level), although some manually built "grammar" has been tried [29] and showing significant improvements. It is actually one of the main theme of this thesis that the grammar learnt from the phoneme symbol can improve the performance of the continuous speech recognition system a lot.

1.4 Research Goal

In this thesis, a decoupled system approach consisting of two modules is adopted. In one module, a recurrent neural network is used to associate acoustic vectors to phoneme symbols as well as capturing long-term contextual information. In another module, the linguistic information of the phoneme sequence is learnt, and is characterized by a finite state grammar. It is to be shown that both sides capture context information well and that they can be trained separately in an efficient way. Since the two modules are decoupled, it is possible to use any other classifier which suits best the task in practical situation.

The use of linguistic information in the symbol level is usually done in a single step together with the association of acoustic vectors to phoneme symbols in most continuous speech recognition systmes (for example, in hidden Markov model) and therefore cannot be trained separately. Besides, it is shown in chapter 4 that usually these approaches are not using the linguistic information at this level (phoneme sequence) well. Therefore a finite state grammar is introduced and described in chapter 5, which is shown to be capturing the "structure" of phoneme sequence very well.

1.5 Thesis outline

This thesis consists of 7 chapters. Excluding **Chapter 1** the introduction and **Chapter 7** the conclusions, this thesis can be roughly divided into 3 parts, with the first 2 describing the pattern classifier module and the context module implemented. The last one is devoted to the integrating of the two modules.

Chapter 2 describes some of the current sucessful approaches to continuous speech recognition, with comments on their pros and cons. As neural network shows a good pattern classification performance, **Chapter 3** describes in detail the use of neural network in continuous speech recognition and some experiments on different architectures of neural network in classifying continuous speech. **Chapter 4 5**) describe the module that deals with the context (linguistic) information in the phoneme sequence. **Chapter 4** introduces the usefulness of context information in the phoneme symbol level and some current approaches which make use of this kind of information is described in detail. **Chapter 5** describes the detail of the finite state grammar where experiments are also carried out to analyze the performance of it. Finally, **Chapter 6** describes the integrated system integrating the neural network and the finite state grammar. Some postprocessing of the neural network is described which is necessary to interface between the two. The final results of the integrated system is also tabulated.

Chapter 2

Current Approaches to Continuous Speech Recognition

2.1 BASIC STEPS FOR CONTINUOUS SPEECH RECOGNITION

A number of approaches to continuous speech recognition have been proposed in the past few years (for example in [22, 26, 32, 29, 2, 38]). Although they adopted different ways to deal with the problems of speech recognition, they all include basically the following steps, as shown in Fig. 2.1.

1. *Signal Processing and Feature Extraction*

   This is the elementary level of all speech recognition system, and is the level which has the smallest variations among all the approaches. At this level, speech signal is converted to some kind of parametric representation which generally has a lower information rate. Before it is converted,

the speech signal has to be preprocessed in order to facilitate the conversion, such as background noise removal and spectral flattening. Different kinds of representation have been studied, but the common ones are zero crossing rates, level crossing rates, short time energy and various types of spectral analysis. Among the various kinds of feature extraction methods, spectral analysis methods are generally used, while the others (very often in time domain, e.g. zero crossing rates) are only used as a supplement in particular application. Some of the spectral analysis techniques used are LPC-derived cepstral coefficients [22, 26], and spectral slope coefficients [29]. As mentioned in section 1.1, the quasi-periodic nature of speech is difficult to represent. Most continuous speech recognition systems now assume that spectral analysis is the best way to extract acoustic features in speech, simply because it outperforms other feature extraction methods.

2. *Segmentation and Pattern Matching*

As mentioned before, sub-word units are usually chosen as the basic recognition units and it is shown in section 1.2 that owing to the constraints of dataset size and feasibility of training, usually context-sensitive subword units are chosen, even though there exists other less context-sensitive units. Even though the acoustic properties of these subword units might be highly variable, it is usually assumed that a mapping of acoustic properties to the subword unit labels exists and can be learned and applied. For the problems mentioned in section 1.3 of the introduction concerning

speaker-independent continuous speech recognition systems, enough training data must be provided to capture the variations of different speakers. It is usually assumed in current continuous speech recognition systems that in spite of the great variations in the acoustic properties of utterances among speakers, there exists some "essential features" in every subword unit symbols that are invariant and can be learnt.

Before the mappings of acoustic vectors to their symbols are learned, the speech signal has to be segmented into discrete region in time so that a label can be attached to that discrete region of speech signal. After segmenting and labeling the speech signal, the acoustic properties, or patterns, that are most representative of the labels have to be extracted, learned and stored. During recognition, the incoming speech are then segmented and the acoustic properties of each segment are compared to the learned patterns of each label, and the label which "matches" best to that segment of speech is then chosen and attached to it.

3. *Word Formation and Syntactic Processing*

After the pattern matching step, a sequence of subword unit labels each corresponding to certain frames of speech is obtained. In order to make this stream of labels understandable, a process has to be taken in order to merge these labels to understandable "words". A dictionary has to be available at this stage which contains all the possible "vocabularies" with their subword symbol transcriptions. The process of merging of subword symbols to the vocabularies is usually called *lexical access.* Since word boundaries are unknown, the number of possibilities of words formed from

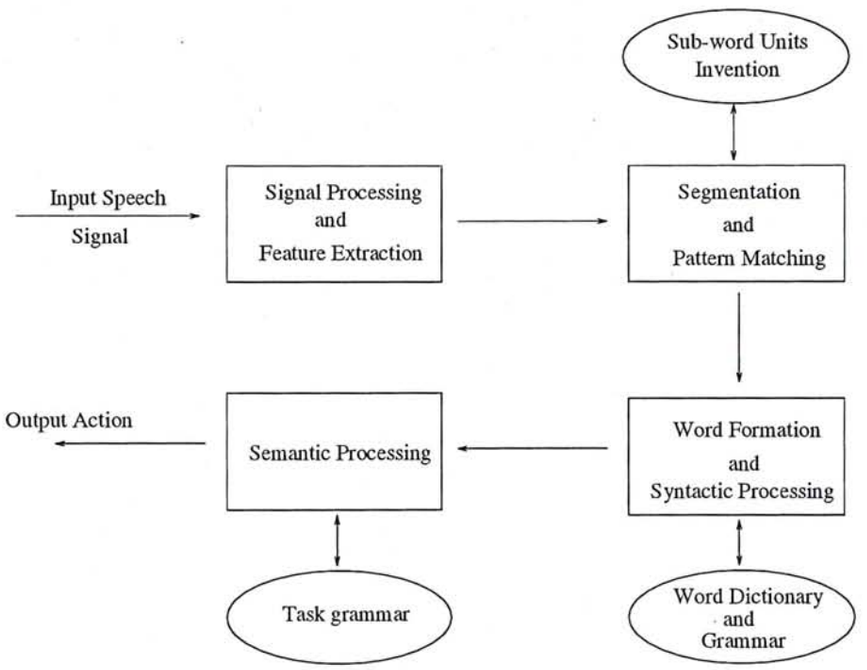


Figure 2.1: Basic steps in a speech recognition system.

the subword symbols in this way are large, and the combinations of words that is invalid according to the rules of English syntax (grammar) have to be ruled out in order to guide the search to obtain the most "probable" sentence. This is the syntactic language model mentioned in section 1.4 of the introduction. Although syntactic processing is important, in actual speech recognition, the reliability of the sub-word labels provided to the lexical access stage is usually a major problem [22].

4. *Semantic Processing*

As mentioned in section 1.4, the use of language model in the semantic level is essential in large vocabulary continuous speech recognition tasks. In this stage, the sentences consisting of sequences of words have to be

understood within the task domain. For example, in a airline ticketing application, the main concern is the flight number, the time, destination ...etc, but the actions to be performed are many, like reservation, flight checking and confirmation..etc. Even though all the essential words are captured, a correction action still have to be determined. The underlying meaning of the words therefore has to be retrieved. In other words, in this stage, the correct action to be performed has to be determined from the recognized words.

Although above are the general common steps used in a speech recognition system, there are a number of different approaches in implementing them. We shall consider some most successful approaches in the last few years in the followings. Since there are common steps among these systems, only the details of the special points in each of these approaches are discussed, instead of describing every "blocks" of the whole systems.

2.2 THE HIDDEN MARKOV MODEL APPROACH

2.2.1 Introduction

One simple way to view the introduction of HMMs is to consider HMMs as a discrete time normalization problem. In dynamic programming (DP), time normalization and pattern matching are accomplished in a single discrete optimization procedure in which the incoming speech signal, decomposed into a

sequence of feature vectors, is matched against pre-computed reference vectors (i.e. a *codebook*). As an example, we can see that in Fig. 2.2, the horizontal axis represents the input utterance with increasing number of frames and the vertical axis represents the phonemes labels. The task of dynamic programming is to find the optimized path which best match the speech signal after time normalization. The arrows in Fig. 2.2 indicates that frame *1* is associated to phoneme label *a*, while frames *2 and 3* are associated to phoneme label *b* and so on.

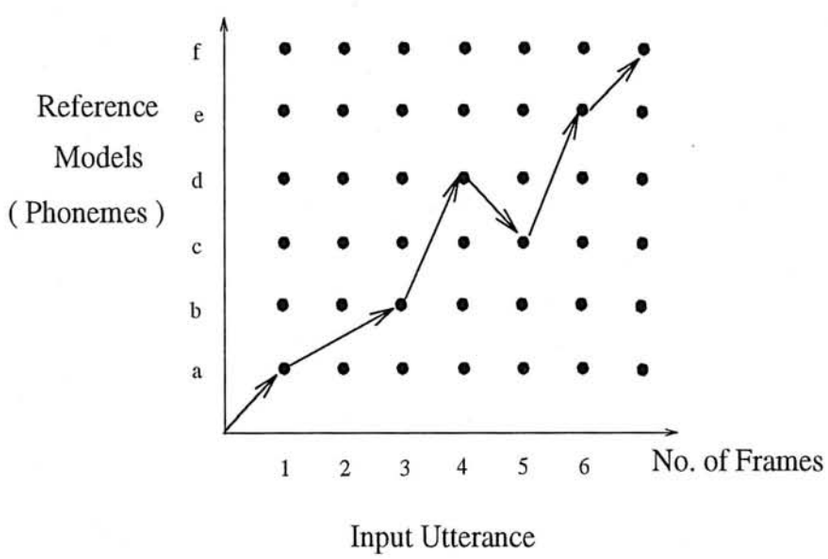


Figure 2.2: Dynamic programming approach to time normalization.

The major advantage of HMM is that a local constraint function can be reestimated, or optimized by an iterative training procedure and there exist algorithms that are efficient enough for the task. In this section, both the "segmentation and pattern matching" and the "word formation and syntactic processing" stages of hidden Markov model are briefly reviewed.

2.2.2 Segmentation and Pattern Matching

1. Models of Sub-word Units

   An example of the basic HMM representation of a subword unit is shown in Fig. 2.3, where the arrows represent the transition from one state to another and the P's are the corresponding probabilities of the transitions. Inside each state, a vector quantization (VQ) codebook is trained to characterize the input speech vectors. If N states are used to represent a subword units, N VQ codebooks are needed. The number of states used to represent a phoneme is constrained by the minimum duration of that phoneme. If the minimum duration of the phoneme is N, then at least N states are needed.

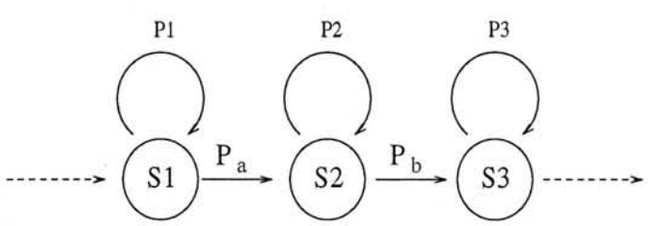


Figure 2.3: The basic HMM representation of a subword unit.

One special thing should be noted about the choice of the subword model is that once the number of states is chosen, it cannot be changed during training. In other words, the training process is searching for a *"maximum likelihood"* of the acoustic vectors given the pre-defined model. The assumption is that the model (with all its assumption relative to its topology and probability density function) is accurate and actually reflects the

structure of the data. The consequence is that both the word formation and language modeling will be dependent on this pre-defined model.

2. Pattern Matching Techniques

   Within each state of the HMM, a vector quantization (VQ) codebook is used[1]. The codebook can either have discrete probability density, continuous mixture probability density or continuous probability density[21]. Fig. 2.4 shows an example of how each of these densities will cover an acoustic space (shown as square).

   For a discrete density VQ, the acoustic space is partitioned into a number of cells with their corresponding cell centroids (codebook vectors). Each centroids is assigned a fixed probability P and an acoustic vector is compared to each of these centroids by some spectral distance measure to determine the closest one. This acoustic vector is then said to have probability P in the closest cell. The continuous mixture probability density VQ uses distinct mixture of Gaussian densities (each with means and variances) for each sub-word unit. The oval boundaries in Fig. 2.4 represents the models for different sub-word units. Since the parameters in each Gaussian density of each model is highly dependent of the characteristics of the model, the models are highly non-overlapping in the acoustic space. The mixed density method or the semi-continuous modeling method [11] tries to cover the entire acoustic space by a set of independent Gaussian

[1]Although VQ might not be the only possible pattern matching technique, it is quoted here as an illustration of the HMM method.

densities. The probability of acoustic vector is calculated by the closeness of the acoustic vector to the codebook vector (i.e. it calculates the exponents of the Gaussian distribution).

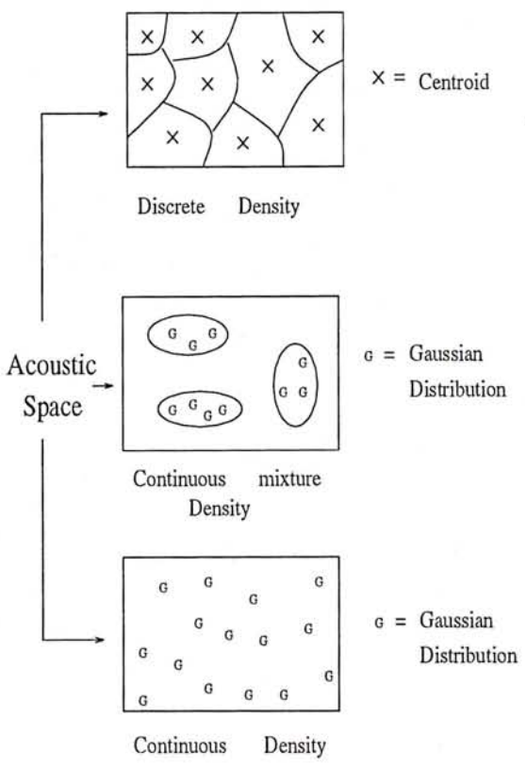


Figure 2.4: Different ways of covering an acoustic space by different densities partitioning.

The training starts by an initial estimation of the means and variances of the VQ and a linear segmentation of the training utterances into the sub-word units and HMM states. Then a better segmentation is obtained by using Viterbi decoding, and the means and variances of the VQs are re-estimated according to the new segmentation. The mathematical details are too lengthy and are not quoted here. The whole process is iterated

until convergence. As mentioned before, the training of the codebook and segmentation of speech frames into states is heavily dependent on the choice of the initial topologies. In other words, although the *"likelihood"* is trained according to the acoustic characteristics of the speech frames, the initial topology is not. Another problem is that the hidden Markov model trained is not discriminative. Each hidden Markov model state only generates a high probabilities for its own class, but has no measure to discriminate against its rivals.

3. Context Dependent Sub-word Units

   One of the major problem of hidden Markov model is that it has poor ability in handling context. Since most pre-defined sub-word units are chosen independent of its context (the effects of preceding and following sounds), the actual temporal and spectral properties of speech is highly dependent on context. The only possible way that context information can be learnt is to redefine the set of subword units by adding some *context dependent* subword units. The use of context dependent subword units is aiming at learning the temporal and spectral properties of certain co-occurrence of context-independent sub-word units. The only change to be made is to change the word lexicon to include the additional context-dependent units. For example, consider the word *"above"*, with the following different representation [27]:

ax	b	ah	v	**Context Independent units**
$-ax-b	ax-b-ah	b-ah-v	ah-v-$	**Triphones (Context dependent)**
ax2	b2	ah1	v1	**Multiple Phone Units**

The word "triphones" is used to signify the use of three context independent sub-word units to represent the concatenation of them. The problem with these kinds of representation is that the number of training samples of these units are small, and most units are seen rarely. Either a small number of these triphone models are used to supplement the context independent units or a huge training set would be needed. The idea of multiple phone units is to cluster common contexts together so as to reduce the number of context dependent models, but this leads to a difficulty in defining lexical entries in the dictionary. The result by [21] shows a 6-7 % improvement in word accuracy by using context dependent units compare to context independent units.

2.2.3 Word Formation and Syntactic Processing

1. Recognition

   The recognition of a speech signal involves the evaluation of the probability of the acoustic vectors of the speech signal according to the trained HMM model. This process involves matching of the acoustic vectors to every state sequence of the HMM. The best state sequence is then selected and the corresponding probability is then given to the speech signal. The simplest method is to calculate the probability through every state sequence, then the highest probable one can be picked out. If there are N distinct

states in the HMM and the speech signal can be decomposed into T acoustic vectors (frames), then it will require computations on the order of $2T \times N^T$. For N = 5, T= 100, then the amount of computation required will be on the order of $\approx 10^{72}$ which is quite infeasible. Fortunately, a more efficient algorithm called the *forward (backward) procedure* exists[27]. The algorithm cut down the computation to the order of $N^2 \times T$. Thus, using the above example, the computation required would be ≈ 3000 only.

The subword units are simply concatenated to form the "whole word models", which are built according to the dictionary. A language model is sometimes used in constraining the search of the most likely state sequence. One of the simple statistical language model is called the bigram probabilities. The bigram probabilities are calculated by counting the frequencies of co-occurences of every pair of words.

2.2.4 Discussion

We see how the HMM can combine the technique of time normalization and pattern matching together in a single step, which in fact is the main advantage of this approach. Besides this elegance of hidden Markov model, it has several drawbacks. In the low level (acoustic pattern matching), it has poor discriminative abilities and depends on the pre-defined model. Also, it assumes that speech signal can be represented as a succession of states, with instantaneous transitions between these states. In a higher level (temporal structure modeling of speech), hidden Markov model can be seen as a stochastic finite state machine (although the topology is not learned from the training data) with a

stochastic output process associated with each state of the finite state machine. Although much success have been obtained for speech recognition using HMM alone, others methods have been proposed to improve the system. In the lower level, these approaches mainly concentrates on improving the relatively week pattern matching techniques used in HMM (the VQ method). In the higher level, the unlearnable topology of the transition states are improved. We are going to discuss the alternative methods at both levels in the next sections.

2.3 NEURAL NETWORK APPROACH

2.3.1 Introduction

As mentioned in the last section, HMM approach uses vector quantization (with multivariate Gaussian density function) to associate a given speech frame to a state in a sub-word unit label. Recently, studies have shown that multilayer perceptrons (MLP) seems to outperform the Gaussian mixture classifier in hidden Markov model, especially in discriminatory ability [36, 28]. A neural network, is basically a dense interconnection of simple, non-linear, computation elements of the type shown in Fig. 2.5. The inputs are summed with weights and then give out the output nonlinearly,

$$output = f(\sum_{i=1}^{N} w_i x_i) \tag{2.1}$$

and the function f is called the activation function of the neurons. The activation function is usually chosen as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

which is called the sigmoid function. The main advantages of using neural network are that firstly, it is ready to be implemented on massive parallel computers, and secondly, because of the nonlinearity within each computational element, a sufficiently large neural network can approximate any nonlinearity of nonlinear dynamical system.

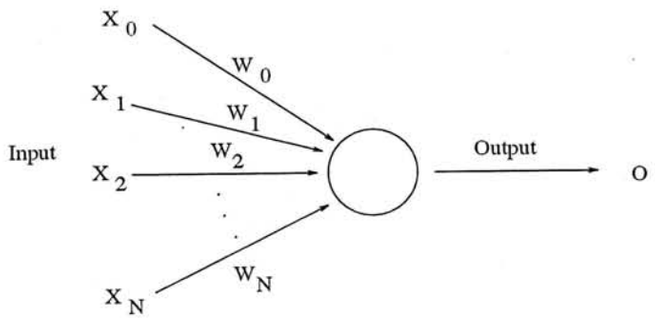


Figure 2.5: The simple computation structure of a neuron in a neural network.

2.3.2 Segmentation and Pattern Matching

1. Training of Neural Networks When MLP is used as a classifier, the input consists of the pattern feature vector and the output usually consists of N nodes, each representing one particular class. Sub-word units are still used as the training units as in other approaches. The basic MLP structure with 3 layers is shown in Fig. 2.6.

   The sub-word units are all labeled in the training set, providing a set of

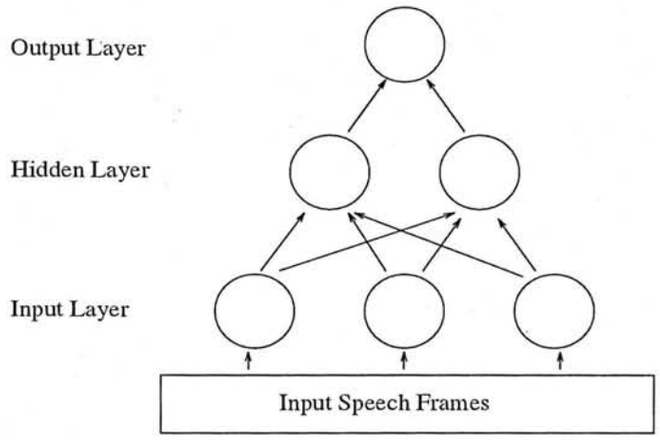


Figure 2.6: An example of an MLP in speech recognition.

input speech frames and a set of coded labels corresponding to this set of speech frames. Then for MLP, iterative training procedures are used to learn the examples of the mapping of the acoustic features of every speech frame to its corresponding label. The most common training procedure is the *gradient descent* learning method, which calculates the mean square errors of all the output from nodes of the MLP and the target, and then improve the weights by sliding downhill on the surface of the weight space. For example, if **E** represents the mean square errors, then

$$E = \frac{1}{2}\sum_{p}\sum_{i}(o_{pi} - t_{pi})^2 \tag{2.3}$$

where o is the input and t is the target, o_{pi} represents the pth training pattern's ith output node. Then

$$w_{ij}(t+1) \;=\; w_{ij}(t) + \Delta w_{ij}(t)$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2.4}$$

where Δw_{ij} represents the change in weight value connecting the i *and* j node and

$$\frac{\partial E}{\partial w_{ij}} = \sum_{p} \frac{\partial E}{\partial o_{pi}} \frac{\partial o_{pi}}{\partial w_{ij}} \tag{2.5}$$

The above equation can then be implemented by putting in the activation function into the derivatives. The weight change can then be calculated both in the input and every hidden layer. In this way, the weights are changed iteratively step by step until convergence and the training patterns are said to be *"stored distributively"* in the weights.

2. Recognition

During recognition, the input speech frames are *"fed"* to the neural network input. The trained weights should be dividing the input hyperspace into distinct classes and each speech frame is then classified to distinct sub-word unit, by comparing the output vector to the pre-determined class output vectors (target). The class that most closely corresponds to the actual output is chosen as the classified output.

2.3.3 Discussion

Although MLP is shown to have a better discriminant power as a classifier, it still has several disadvantages. The most important one concerning speech recognition is the poor ability to handle the speech dynamics, since conventional

artificial neural networks are structured to deal with static patterns. In fact, most successful speech recognition task employing neural networks alone are confined to small vocabulary word recognition or isolated word phoneme recognition. The neural network has no memory element from one classification to the next one, and successive classifications may be contradictory. In addition, the problem of context is unsolved by adopting the approach of neural networks. Although it is possible to include adjacent frames along with the original central frame to feed into the MLP, the dimensionality expansion quickly results in difficulty in obtaining good models of the data.

2.4 MLP/HMM HYBRID APPROACH

2.4.1 Introduction

As we have seen from the previous two sections, the HMM and neural networks approaches each have their own advantages. While HMM has powerful dynamic time warping capabilities but is poor in pattern matching , Neural Networks have poor ability to include dynamics of speech but are good at pattern discriminations. Therefore researches had been directed towards using a hybrid system combining HMM and neural networks [2, 19, 4, 29] in speech recognition. These systems shows a lot of improvement compared to the use of either the HMM or the neural networks alone.

The segmentation and pattern matching step is basically identical to the neural network approach mentioned in section 2.3 which the word formation and syntactic processing step is carried out by the method used in the hidden Markov

model approach mentioned in section 2.2.

2.4.2 Architecture of Hybrid MLP/HMM Systems.

The basic architecture of the MLP/HMM hybrid system often uses the MLP outputs as the probability estimator [2] of associating a particular speech frame to a certain subword unit label (phoneme). The estimated probabilities are then used in the Viterbi search to determine the best time-warped succession of states in the HMM to explain the observed speech measurements. In other words, the MLP is used to estimate probabilities and the HMM is used to incorporate them to segment continuous speech into a succession of words. The basic architecture is shown diagrammatically in Fig. 2.7.

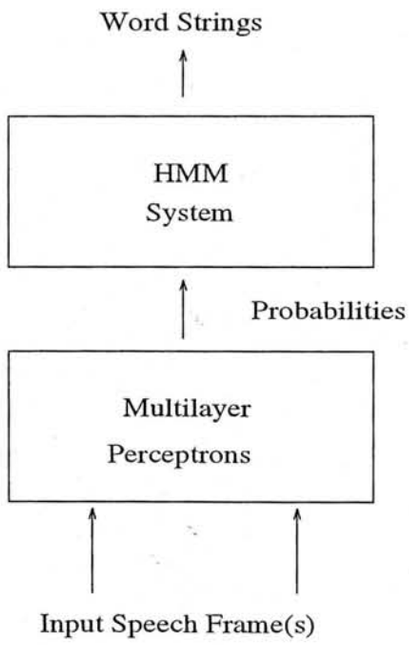


Figure 2.7: The basic architecture of an MLP/HMM hybrid system.

2.4.3 Discussions

There are other related hybrid approaches which use different neural network topologies to obtain the probabilities for the HMM, such as [3, 13] which uses Time Delay Neural Network (TDNN) and Learning Vector Quantization (LVQ) respectively to replace the MLP to suit their own applications. The current works in these areas include the use of different training strategies and different topologies to obtain a better probability estimation, the use of dynamic neural networks to incorporate time dependencies in probability estimation and the use of the *top N* neural network output activations to better utilize the information embedded in the neural network, although they have to be trained together and cannot be separated.

2.5 SYNTACTIC GRAMMAR

2.5.1 Introduction

When dealing with the temporal structure of speech utterance, the relation between speech units is important. Syntactic pattern recognition makes use of the structure of patterns for classification and description. This could be accomplished by defining suitable and distinct *"grammars"* that reflect the structure of each pattern class. Syntactic grammar in continuous speech recognition is usually used at the word formation and syntactic processing stage.

So far, the problem of context information is dealt with at the lower level by using multiple speech frames as the input, so that variations between successive frames can be learnt. This is in fact shown to be successful [2], but again,

compromise has to be made between computation resources and accuracies. In fact, context information should be important at all levels. It is to be shown that besides harnessing the context information in the acoustic level, an alternative is to make use of context information at a higher level (symbol level). One of the approach is to use syntactic grammar to capture the high level context information.

2.5.2 Word formation and Syntactic Processing

In speech recognition, phoneme symbols are concatenated to form a word. Since there exists redundancies in natural spoken English and phoneme sequences are limited either because of the characteristics of the language or the impossibilities of uttering certain phoneme sequences, there exists certain structure in the phoneme sequence that can be learnt from the training samples. This process is generally called *grammar inference.* A syntactic grammar can be inferred from a set of training data consisting only of words and its phonemic transcriptions, disregarding its acoustic properties. After that, any low level pattern recognizer can be used to classify the speech frame and then fed to the syntactic grammar to generate a sequence of words.

A two level grammar can be built where the structures of phonemes *and* words are learnt and stored by the grammar. During recognition, the labels obtained from the pattern recognizer can then be processed by the grammar to form possible words and word sequences.

2.5.3 Discussion

The main advantage of using syntactic grammar is that it does not depend on the acoustic characteristics of the speech vector and can be trained separately from the pattern recognizer. Besides, a grammar can characterize the structural pattern from the training data. Compared to hidden Markov model, the states (or nodes) inside a grammar are learnt from the training data instead of predefined as in hidden Markov model. The transition probabilities between 2 states in hidden Markov model is basically trained by the frequency of co-occurrences of these 2 states, while in a grammar, it is based on the overall structure of the whole word (depends on all previous states). The ability of the syntactic grammar in dealing with the temporal nature speech in the symbol level should therefore be better than hidden Markov model. The detail idea of the syntactic grammar is to be discussed in later chapters.

2.6 SUMMARY

We have briefly reviewed the current approaches to speech recognition. Basically, each of these approaches has its way to tackle the problems mentioned in the introduction. The HMM has good ability to deal with the problem of *dynamic time warping*, and capture very well different features of the dynamically changing nature of speech segments. On the other hand, neural networks have very good ways to create a non-linear mapping between the acoustic features and the sub-word units classes. The discriminating power between classes is also better than the use of HMM alone [2]. The problem of neural network is the poor ability to handle dynamics, but attempts have already been made to

deal with this weak point and the combinations of HMM and neural networks therefore seems to be a natural choice. The use of syntactic grammar has also shown its ability in capturing high level context.

Chapter 3

Neural Network As Pattern Classifier

3.1 INTRODUCTION

Artificial neural networks can be used for tasks such as feature extraction, prediction and in applications ranging from signal processing to stock market forecast. In speech recognition, they are mainly used as a pattern classifier. To be specific, artificial neural networks usually deals with the problem of classifying acoustic vectors into (phonemes) classes. We shall consider the use of several topologies i.e. how the computational elements are interconnected, of artificial neural networks in classifications in continuous speech recognition.

3.2 TRAINING ALGORITHMS AND TOPOLOGIES

A common element among various kinds of artificial neural networks is that they all involves large interconnected networks of relatively simple and typically non-linear units, and the main entities that characterize an artificial neural networks are: (1) its strategy for pattern learning or training, (2) the characteristics of the individual units (neurons) and (3) the ways they are interconnected (topologies). In this section, four popular kinds of neural network are introduced and their basic training algorithms and topologies are briefly reviewed, they are (1) multilayer perceptrons, (2) recurrent neural network, (3) self-organizing mapand (4) learning vector quantization.

3.2.1 Multilayer Perceptrons

An N-layered multilayer perceptrons consists of N+1 layers, each of them contains several units called artificial neurons. These artificial neurons are computational units and their output values are determined by first summing all of their inputs and then passing the results through the sigmoid function f:

$$f = \frac{1}{1 + e^{-x}} \tag{3.1}$$

If $\vec{z}_n(\vec{x})$ denotes the vector of the output values of the (n+1)th layer and $\vec{x}$ is the input vector then

$$\vec{z}_n(\vec{x}) = f(\mathbf{W} \cdot \vec{z}_{n-1}(\vec{x})) \tag{3.2}$$

where **W** is a matrix with size equal to the no. of units in (n-1)th layer $\times$ no. of units in nth layer. The weight matrix **W** are obtained from a set of training inputs and associated output pairs $\vec{d}(\vec{x})$ by minimizing, in parameter space, the error criterion defined as,

$$E = \frac{1}{2}\|\vec{z}_n(\vec{x}) - \vec{d}(\vec{x})\|^2 \tag{3.3}$$

If there are more than 1 layer, $\vec{z}_n(\vec{x})$ is a non-linear function of the input vector $\vec{x}$. The weight matrix is interactively updated via a gradient connection procedure to reduce the error. The basic steps of learning are:

1. Apply input vector to the input layer.
2. "Feed forward" the input vector to determine outputs in all units in all layers.
3. At output layer, calculates the error given by Eq. 3.3.
4. "Back-propagate" error through the network (starting at the output layer).
5. Adjust the weight matrix as:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\frac{\partial E}{\partial w_{ij}(t)} \tag{3.4}$$

where α is the learning step size (learning rate).

Now consider the partial derivative of the error function. By chain rule, we have

$$\begin{aligned}\frac{\partial E}{\partial W_{ij}(t)} &= \frac{\partial E}{\partial act_{n_j}}\frac{\partial act_{n_j}}{\partial W_{ij}(t)} \\ &= -\delta_{n_j} z_{(n-1)_j} \end{aligned} \tag{3.5}$$

where

$$\delta_{n_j} \triangleq -\frac{\partial E}{\partial act_{n_j}},$$

$$z_{n_j} = f(art_{n_j}),$$

$$act_{n_j} = \sum_i w_{ij} z_{(n-1)_i} \tag{3.6}$$

From Eq. 2.4, we have

$$\Delta w_{ij} = \eta \delta_{n_j} z_{n_j} \tag{3.7}$$

Eq. 3.7 is called the *delta learning rule*. We see that the δs can be calculated at each level, starting from the output layer. At the output layer, we have

$$\delta_{n_j} = (d_j - z_j) f_j'(act_{n_j}). \tag{3.8}$$

where f'(x) denotes the first derivative with respect to x.

For units not in the output layer, from Eq. 3.6

$$\delta_{k_j} = -\frac{\partial E}{\partial z_{k_j}} \frac{\partial z_{k_j}}{\partial act_{k_j}} \tag{3.9}$$

$$\frac{\partial E}{\partial z_{k_j}} = \sum_s \frac{\partial E}{\partial act_{(k+1)_s}} \frac{\partial act_{(k+1)_s}}{\partial z_{(k+1)_j}} \tag{3.10}$$

$$= \sum_s (-\delta_{(k+1)_s} w_{(k+1)s}) \tag{3.11}$$

therefore

$$\delta_{k_j} = f_k'(act_{k_j}) \sum_s w_{ks} \tag{3.12}$$

In this way, all the weights can be updated after a training pattern has been feed forward and then backward. For classification purpose, each output node can represent a class and the target vector of that class can be set to one and the rest to zero. In this case, we see that Eq 3.3 can be written as,

$$\begin{aligned} E_T &= \|\vec{z}_n(\vec{x}) - \vec{d}(\vec{x})\|^2 \\ &= \sum_{k=1}^{N} y_k^2(\vec{x}) + 1 - 2y_T(\vec{x}) \end{aligned} \tag{3.13}$$

for input vector $\vec{x}$ belonging to class $\mathbf{T}$. Now since only the term $y_T(\vec{x})$ depends on class $\mathbf{T}$, $y_T(\vec{x})$ (the activation value of the T-th output node of the multilayer perceptrons) can be an indication of how close the input vector $\vec{x}$ is to class $\mathbf{T}$. As $y_T(\vec{x})$ is usually a non-linear function of the n-dimensional input pattern vector $\vec{x}$, a set of non-linear function with numbers equal to the numbers of classes can be obtained after training. An unknown input vector can then be classified according to this set of non-linear functions.

One modification of the traditional multilayer perceptrons suggested in [16] is to connect the input layer to the output layer as well as to the hidden layer, as shown in Fig. 3.1, which is named as *skip connections*. The philosophy behind the use of skip connections is that most classification problems have a significant linear component in their solutions and the connection of input to output units places the linear component of the solution in those connections while the non-linear component is isolated in the hidden layer. In [34], it is shown that without skip connections, more nodes are needed in the hidden layer to account for the linear component of the solution.

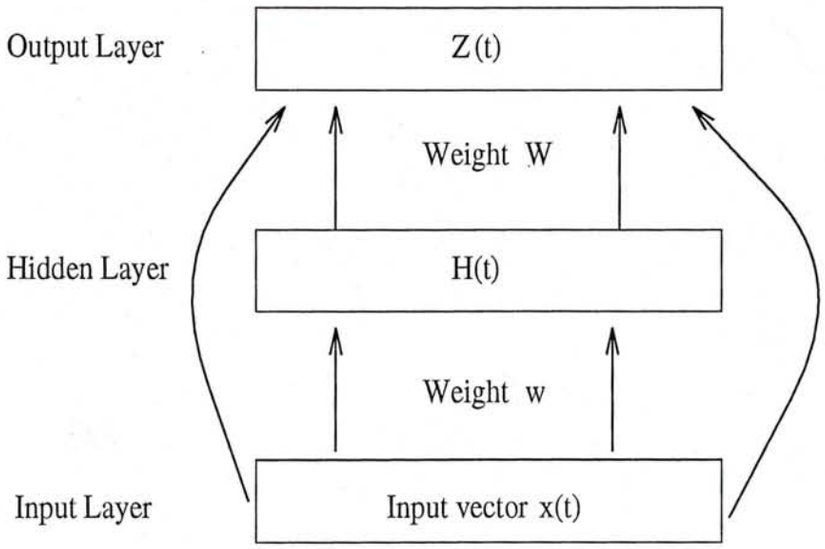


Figure 3.1: Use of skip connections in multilayer perceptrons.

3.2.2 Recurrent Neural Networks

We have seen in the last section how artificial neural networks can be used as a pattern classifier. While it was shown that multilayer perceptrons can form arbitrary non-linear decision surface (with sufficient hidden units), the conventional structure mentioned can only deal with static patterns. Since speech is inherently dynamic in nature, some modifications in the structures of the multilayer perceptrons is required.

In speech recognition, the classification of a speech frame will be more accurate if we can account for what we saw at earlier times. This is especially true in continuous speech recognition. Since continuous speech is usually divided into labeled discrete frames before recognition, the relation between consecutive frames is strong owing to the many slowly varying contextual variables such as the vocal tract length, speaker rate, background noise and channel distortion.

Hence, efficient use of these context information should increase the performance of the speech recognition systems.

The most popular tool for this task is the hidden Markov model [22, 26, 32, 20]. As we have seen in section 2.2, hidden Markov model assumes that a speech frame is dependent purely on a particular hidden Markov model state within a particular phoneme class. However, there are a lot of contextual variables which affect the characteristics of the speech frames and cannot be described entirely by a function of the current state of the phoneme class alone. Methods are required which has the ability to capture higher order correlations over long time periods.

Consider a recurrent neural network with topologies shown diagrammatically [16] in Fig. 3.2.

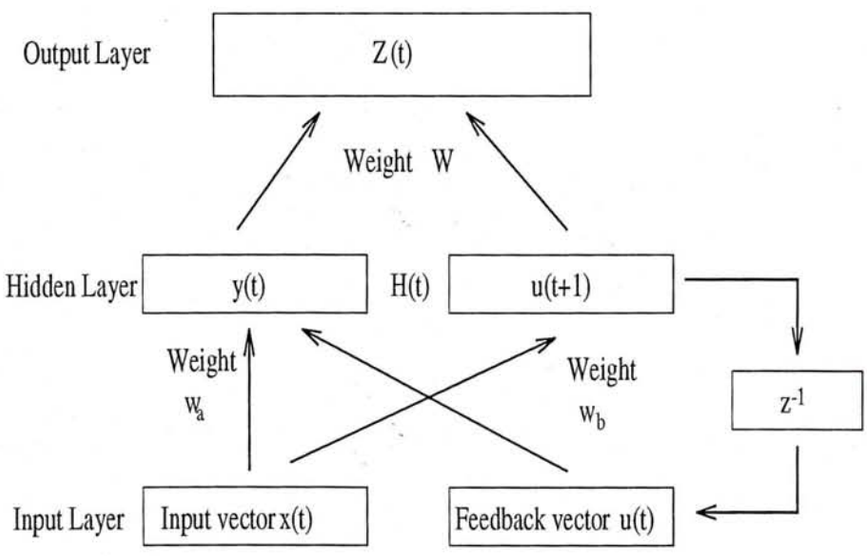


Figure 3.2: A recurrent neural network.

If we use f(x) to denote the activation function of each computational neurons, then

$$\vec{Z}(t) = f(\vec{W} \cdot \vec{H}(t)) \tag{3.14}$$

where

$$\vec{H}(t) = \begin{bmatrix} f(\vec{w_a} \begin{bmatrix} \vec{x}(t) \\ \vec{u}(t) \end{bmatrix}) \\ \vec{u}(t+1) \end{bmatrix} \tag{3.15}$$

$$\vec{u}(t+1) = f(\vec{w_b} \begin{bmatrix} \vec{x}(t) \\ \vec{u}(t) \end{bmatrix}) \tag{3.16}$$

The recurrent neural networkin Fig 3.2 is now able to learn the sequence mappings of infinite duration. The training steps are similar to those of the static multilayer perceptrons, except that the initial value of $\vec{u}(0)$ have to be preset. A reasonable choice would be setting all the elements in $\vec{u}(0)$ to be ones. Sigmoid function can still be used as the activation function and backpropagation learning is carried out as in Eq 3.5.

3.2.3 Self-organizing Maps

In [18], the author has shown an alternative neural learning structure which perform dimension reduction through the conversion of feature space to yield a *topologically ordered* similarity maps. The "neurons" in a self-organizing map is a weight vector with dimension equal to the input vectors. The topology commonly used is usually 2-dimensional, yielding a planar map as in Fig 3.3.

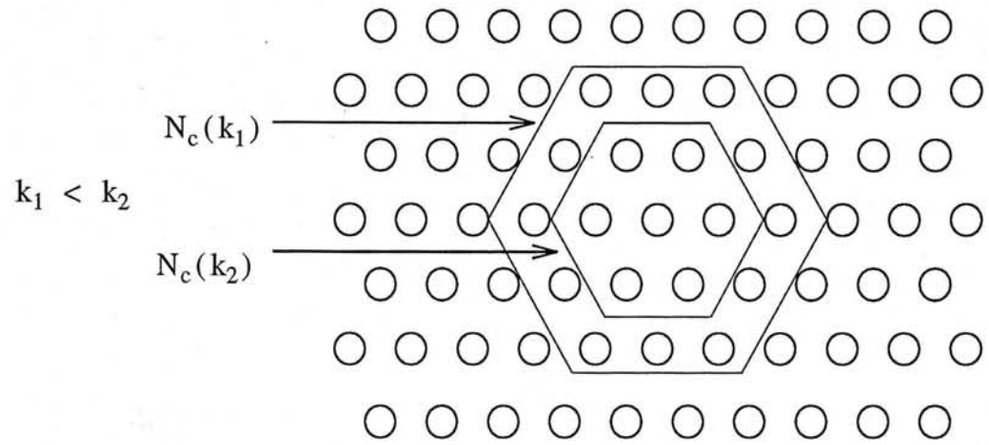


Figure 3.3: A 2-D self-organizing map. with time-varying neighbourhood function N_c.

The learning process is to obtain individual neural clusters to reflect the pattern similarity in the training data set. Initially, the weights of all units are randomly chosen. Then the Euclidean distances between the input vector and all the weight vector are computed. The weight vector with the minimum distance to that input vector is called the "winner". Now, if $w_c(k)$ represents the winning weight vector at the kth input vector, the updating rule is:

$$\vec{w_i}(k+1) = \begin{cases} \vec{w_i}(k) + \alpha(k)[\vec{x}(k) - \vec{w_i}(k)] & i \in N_c(k); \\ \\ \vec{w_i}(k) & i \notin N_c(k); \end{cases} \tag{3.17}$$

where $N_c(k)$ is the set of neurons neighbouring around the winning neuron. The neighbourhood size of $N_c(k)$ is usually defined in terms of the topological distance of the neurons to the winning neuron, and can be varying with time during training.

The main characteristics of a self-organizing map can be observed from the updating rule. It can be seen that the distance between the input and the updating weight vectors inside N_c are decreased. The resultant map is a map of trained weight vectors with each of them having neighbouring weight vectors relatively closer to itself than the weight vectors further away.

3.2.4 Learning Vector Quantization

The learning vector quantization [24] is similar to the k-means clustering algorithm[1] and the self-organizing map mentioned in the last section. A set of weight vectors with dimension equal to the input vector is chosen at first. This set of vectors are assumed to be a set of labeled reference vectors each representing a class of patterns. This set of reference vectors are usually chosen from the training set initially, then after an input vector is presented, a weight vector closest to this input vector is computed. If the winning vector is denoted by μ_c, then the weight vectors are updated during the training phase as,

$$\begin{aligned} \mu_c(t+1) &= \begin{cases} \mu_c(t) + \alpha(t)[x(t) - \mu_c(t)] & \text{if x is correctly classified,} \\ \mu_c(t) - \alpha(t)[x(t) - \mu_c(t)] & \text{if x is misclassified.} \end{cases} \\ \mu_i(t+1) &= \mu_i(t) \qquad \text{if i} \neq \text{c.} \end{aligned} \tag{3.18}$$

Therefore a correct classification needs a refinement of μ_c towards the input vector and misclassified push μ_c to the opposite direction in the vector space. The reference vectors obtained will be a number of various clusters characterizing

the training data.

The current problems of the hybrid HMM/MLP system is that the training time needed for the neural network is rather long (several days in a fast transputer [29]). It is shown that increase in network size actually can yield a better accuracy in the neural classifier [2, 29], but when resources are limited, a compromise has to be made.

3.3 EXPERIMENTS

3.3.1 The Data Set

The database used is the DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus. The TIMIT database is a large database containing a total of 6300 sentences. 10 sentences spoken by each of the 630 speakers from 8 major dialect regions of the United States. Owing to limitations of memory and computational resources, only one dialect region is selected for the experiment. The SX sentences in the database are selected which "were designed to provide a good coverage of pairs of phoneme, with extra difficulties or of phonetic contexts thought to be either difficult or of particular interest" according to the description of the database. In the training set, all of the 49 speakers (31 male, 18 female) from the dialect region dr1 (New England) with each speaker uttering 1 sentence is used. The testing set comprises of all the sentences in the "core test set" of the same dialect region. The core test set is designed such that no text utterance is identical to any of the training set. The core test set contains 15 sentences uttered by 3 speakers (2 male, 1 female), with each of

them uttered 5 sentences. All the sentences are already labeled by 61 different phoneme symbols. The symbols used are shown in Tab. 3.1.

Phn	No.	Phn	No.	Phn	No.	Phn	No.	Phn	No.	Phn	No.
b	1	sh	12	en	23	ih	34	uh	45	dcl	56
d	2	z	13	eng	24	eh	35	uw	46	gcl	57
g	3	zh	14	nx	25	ey	36	ux	47	pcl	58
p	4	f	15	l	26	ae	37	er	48	tck	59
t	5	th	16	r	27	aa	38	ax	49	kcl	60
k	6	v	17	w	28	aw	39	ix	50	tcl	61
dx	7	dh	18	y	29	ay	40	axr	51		
q	8	m	19	hh	30	ah	41	ax-h	52		
jh	9	n	20	hv	31	ao	42	pau	53		
ch	10	ng	21	el	32	oy	43	epi	54		
s	11	em	22	iy	33	ow	44	bcl	55		

Table 3.1: Table of the phoneme symbols used in the experiments

3.3.2 Preprocessing of the Speech Data

The original speech data is sampled at 16kHz. After preprocessing, a number of feature vectors are obtained. The overall procedures of the preprocessing can be shown as a block diagram in Fig. 3.4

1. *Pre-emphasis*

   The speech signal is passed through a first order digital network to spectrally flatten the signal. The filter used in the experiment is:

$$H(z) = 1 - 0.97z^{-1} \tag{3.19}$$

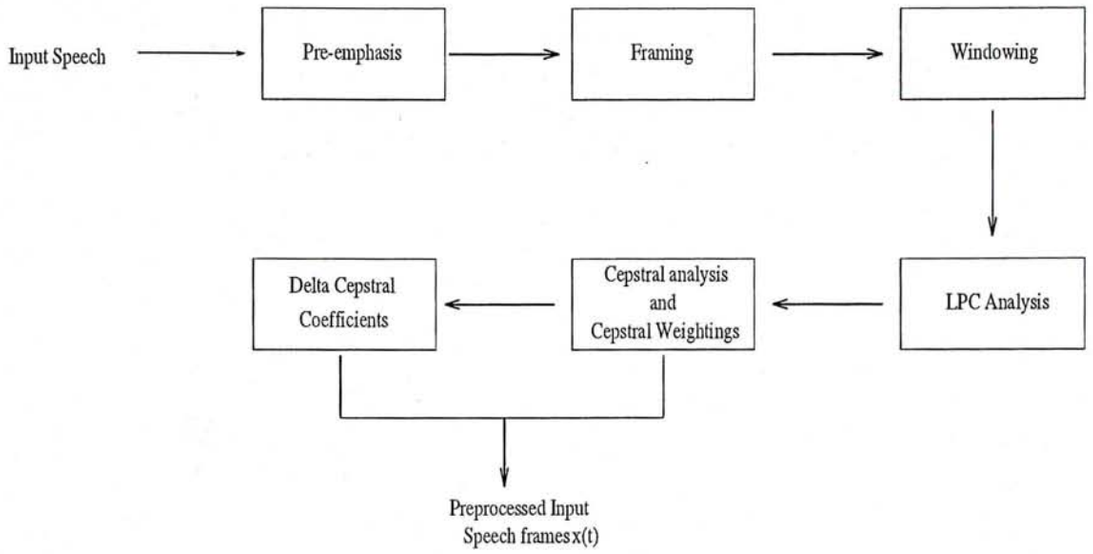


Figure 3.4: The basic blocks of the preprocessing of speech data.

2. *Framing the speech samples*

   Every 256 samples of speech is blocked to a speech frame. Therefore each frame lasts 16ms ($256 \times \frac{1}{16}$). Consecutive frames have 8ms overlap, as shown in Fig. 3.5. Overlapping frames have the advantage of increasing correlations between frames and the spectral parameters will be more "smooth".

3. *Windowing*

   In order to minimize the adverse effects of discontinuities caused by chopping a section out of the running speech signal, usually a window function is chosen such that it can taper the signal to zeros at the beginning and end of each frame. A Hamming window function w(n) is used here where

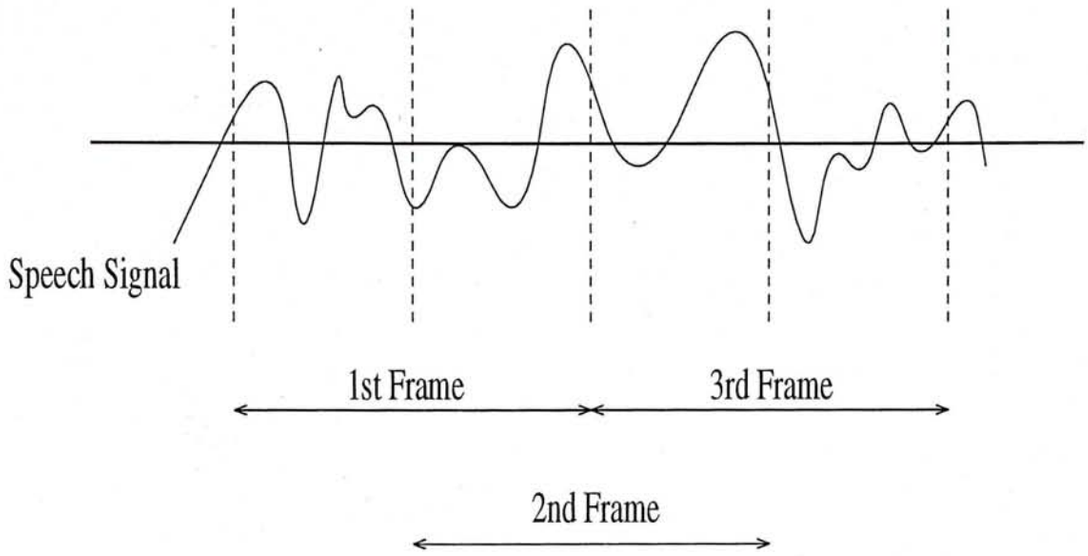


Figure 3.5: The overlapping of speech frames.

$$w(n) = 0.54 - 0.46 \times \cos(\frac{2\pi n}{N-1}) \qquad 0 \leq n \leq N-1 \qquad (3.20)$$

where N is the number of samples inside the a speech frame.

4. *LPC analysis*

A linear predictive coding (LPC) coefficient vector is computed from the windowed speech frame. First the autocorrelation coefficient $r_l(m)$' is computed by

$$r_l(m) = \sum_{n=0}^{N-1-m} x_l(m)x_l(n+m) \qquad m = 0, 1, \ldots, p. \qquad (3.21)$$

where x(n) is the nth sample of the speech frame and p is the order of LPC to be calculated. Here we choose p=16. A larger p increases the spectral

resolution of the signal, in the expense of computations. From the auto-correlation coefficients, a classical method called the Durbin's method [27] can be used, which can be represented briefly by the following algorithm,

$$\begin{aligned} E^{(0)} &= r(0) \\ k_i &= \frac{r(i)-\sum_{j-1}^{p-1}\alpha_j^{i-1}r(|i-j|)}{E^{(i-1)}} \qquad 1 \le i \le p \\ \alpha_i^i &= k_i \\ \alpha_j^i &= \alpha_j^{i-1} - k_i\alpha_{i-j}^{i-1} \\ E^{(i)} &= (1-k_i^2)E^{(i-1)} \end{aligned} \tag{3.22}$$

By solving the equation iteratively, we obtain the LPC coefficients a_n as

$$a_n = \alpha_m^{(p)} \tag{3.23}$$

5. *Cepstral analysis and cepstral weighting*

The LPC derived cepstral coefficients are shown to be more robust, reliable feature representation for speech recognition than the LPC coefficients [15]. The conversion is done by the recursion,

$$c_m = a_m + \sum_{k=1}^{m-1}\frac{k}{m}c_k a_{m-k}. \qquad 1 \le m \le p. \tag{3.24}$$

$$c_m = \sum_{k=1}^{m-1}\frac{k}{m}c_k a_{m-k}. \qquad m > p. \tag{3.25}$$

Because of the sensitivity of the low-order cepstral coefficient to overall spectral slope and the sensitivity of the higher order cepstral coefficient to noise [15], the cepstral coefficients are tapered by a window function so as to minimize these sensitivities. The window function is calculated by,

$$w_c(m) = 1 + \frac{p}{2}\sin(\frac{\pi m}{p}) \qquad 1 < m < p. \tag{3.26}$$

to give

$$\hat{c}(m) = c(m)w_c(m). \tag{3.27}$$

6. *Delta cepstral coefficients*

Delta cepstral coefficient is an improved representation from the cepstral coefficient by including the first order derivative of the cepstral coefficients to capture the temporal order of the spectrum [8]. The derivative is approximated by

$$\frac{\partial c_m(t)}{\partial t} = \Delta c_m(t) \approx \mu \sum_{z=-k}^{k} c_m(t+k). \tag{3.28}$$

where μ is a normalization constant. k is chosen to be 3 in the experiment, which is shown to be a good estimate of the derivative in [8].

The resultant vector x(t) has 32 elements (16 cepstral coefficient + 16 delta cepstral coefficient),

$$x(t) = (\hat{c_1}(t), \hat{c_2}(t), \ldots, \hat{c_{16}}(t), \ldots, \Delta\hat{c_1}(t), \ldots, \Delta\hat{c_{16}}(t)) \tag{3.29}$$

Each of the input vector is labeled by the given phoneme symbol labeling in the original database.

3.3.3 The Pattern Classifiers

1. *Static multilayer perceptrons*

   The 32-dimension input vector is fed to the input layer frame-by-frame. The phonemes symbols are coded by a number as shown in Tab. 3.1. The target vector for each speech frame is a 61-dimension vector with the element of the correct phoneme class set to one and the rest to zeros. The target vector of the "silenced" speech frame is set to all zeros. Backpropagation training is done according to Eq 3.5 in section 3.2.1. A momentum term of 0.9 is introduced. A learning rate of 0.01 is chosen which steadily decrease to 0.001 as the training epochs go. The partial derivatives are summed every 2000 frames to give the estimate of the gradient. Theoretically, the more the number of frames accumulated before updating, the more accurate the estimate should be, but less weight updatings are carried out. It was shown in [35] that the solution converged should be the same.

2. *Recurrent neural network*

   The recurrent neural network used has the same architecture [29] as shown in Fig. 3.2. The feedback vector has a dimension of 200. The initial feedback vector is all set to one. The same set of parameters in static multilayer perceptrons is used here.

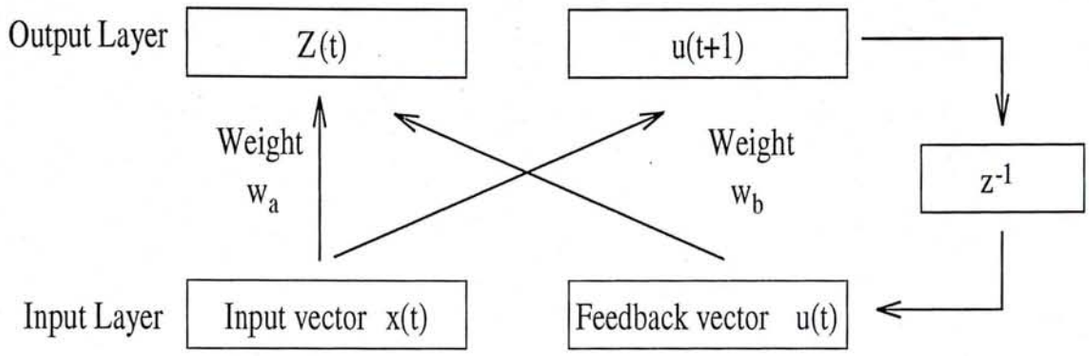


Figure 3.6: The recurrent neural network without hidden layer.

A recurrent neural network without hidden layer is tried also. The architecture can be shown diagrammatically as in Fig. 3.6. One special thing is that since there is no target value for the feedback vector, the weights w_a shown in Fig. 3.6 is not trained and therefore depends very much on its initialization. No attempt was made to investigate the choice of the initial weight values, though it is believed to be worth investigating. The output vector Z(t) is now updated as,

$$\vec{Z}(t) = f(\vec{w_a} \begin{bmatrix} \vec{x}(t) \\ \vec{u}(t) \end{bmatrix}) \tag{3.30}$$

$$\vec{u}(t+1) = f(\vec{w_b} \begin{bmatrix} \vec{x}(t) \\ \vec{u}(t) \end{bmatrix}) \tag{3.31}$$

3. *Self-Organizing Map*

The self-organizing map is trained by the updating rule stated in section 3.2.3 with topology indicated in Fig. 3.3. Initially, the reference vectors of the map are initialized randomly. During training, a Gaussian neighbourhood function $h_c(t)$ is used, where

$$h_c(t) = \alpha(t) - exp(\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}) \tag{3.32}$$

and $\alpha(t)$ is the learning rate and $\sigma(t)$ is the radius of $N_c(t)$. Initially, the radius is chosen as the size of the map in order to make those reference vector to be "ordered". The learning rate is chosen as 0.1 initially. Both the learning rate and radius decreases linearly to zero when training ends. The second stage of training is then carried out to "fine-tune" the weight vectors. The radius is chosen initially as half the map size and the learning rate as 0.01. Both are linearly decreased to 0 as training goes on. The evaluation of the map is done by a majority vote of the k-nearest weight vector to the testing input vector.

4. *Learning Vector Quantization*

The learning vector quantization map is trained by the updating rule stated in section 3.2.4. Initially, a same number of codebook vectors are chosen from the training set for each class of phoneme. The medians of the shortest distance for each class is calculated by using the training set to do a k-nearest neighbour classification. If the medians of the shortest distance is above average, a vector is added to the codebook from the training set. This is done to make a better estimates of the number of clusters for every class. If the medians of shortest distance is below average, some codebook

vectors are deleted from those classes. The initialized codebook can then be used for training by Eq. 3.18. Evaluation is simply done by finding the minimum distance codebook vectors to the input testing vector to see if their labels match.

3.4 RESULTS AND DISCUSSIONS

Tab. 3.2 gives the accuracies of the multilayer perceptrons and recurrent neural network classifier by taking the maximum activation (top 1) and 5-top highest activations (top 5). A choice of 200 hidden units is used to show the effect of including non-linear mappings. The training is stopped by a cross-validation strategy, in which training stops whenever the root mean square error of the testing data starts to increase again.

MLP	No hidden units		200 hidden units	
	Top 1	Top 5	Top 1	Top 5
Training	7.46%	10.25%	13.24%	18.94%
Testing	5.29%	7.29%	8.48%	10.39%

RNN	No hidden units		200 hidden units	
	Top 1	Top 5	Top 1	Top 5
Training	12.07%	16.15%	29.47%	35.29%
Testing	6.94%	8.25%	22.57%	28.32%

Table 3.2: Results of the multilayer perceptrons and the recurrent neural network.

From the results of the static multilayer perceptrons, we see that inclusion of

hidden units has almost the same effect as inclusion of feedback units, although the former shows a better generalizations as shown in the testing. The number of free parameters in the recurrent neural network with no hidden units ($232 \times 61 = 14152$) is also smaller than the number of free parameters in the static neural network with 200 hidden units ($32 \times 200 + 200 \times 61 = 18600$). The inclusion of context "state" vectors shows a much better performance in neural network with and without hidden layer, but improvement by the hidden units seems more significant in the recurrent neural network. We can also see the effects of including the *top 5* output symbols, which significantly increases the accuracies.

Tab. 3.3 shows the accuracies of the learning vector quantization classifier and the self-organizing map, with different number of codebook vectors. For the self-organizing map, different values of k are used during evaluation.

LVQ	1000 codevectors	2000 codevectors
Training	15.20%	22.08%
Testing	6.07%	16.81%

SOM	30×30 codevectors		45×45 codevectors	
	k = 5	k = 10	k = 5	k = 10
Training	18.83%	19.89%	21.99%	22.15%
Testing	16.43%	17.26%	17.11%	17.57%

Table 3.3: Results of learning vector quantization and self-organizing map.

The number of codebook vectors in learning vector quantization and self-organizing

map is almost the same, although the time for training the learning vector quantization is a bit shorter than the training time for the self-organizing map. The change in the value of k in the k-nearest neighbour test doesn't show significant improvements in the self-organizing map. Basically, both pattern recognizer show similar results in both training and testing but self-organizing map seems to outperform the learning vector quantization in the smaller codebook (fewer number of free parameters), probably due to the insufficient codevectors and thus not enough clusters for each class in the learning vector quantization codebook. The 2000 codevectors show a comparable result to the self-organizing map.

This chapter has shown the performance of neural network in the classifcation of phonemes in continuous speech. The use of recurrent neural network to capture context information shows the best performance in all of the classifier tried. Common errors are mislabeling of phonemes or misplacing boundaries between phonemes, but these kind of decision is itself difficult, where no formal rules are availble and the "correct" answer might be subjective. Since the number of free parameters in multilayer perceptrons is larger than recurrent neural network, the use of feedback units is shown to be better than non-linear units when classifying speech frames. Thus, instead of using very complex non-linear mapping function to classify speech frames, more efforts should be paid in making use of the context of speech frames. Later in the thesis, a further use of context information in the higher level is also introduced...

Chapter 4

High Level Context Information

4.1 INTRODUCTION

As mentioned in chapter 1, the compatibility of acoustic and phonetic properties is essential in speech recognition. As we have seen in the previous chapters, speech signal is usually represented by a sequence of measurement vectors and it is difficult even for an phonetician expert to say *what was being said at instant* t . When we use a sequence of symbols (phoneme) to represent the speech utterances, we are actually abstracting the explanation of the acoustic evidence. Each symbol is an object which stands for a structure of organized behaviour which is constituted by more or less arbitrary number of speech frames, though we have given the sequence of symbols very distinct meanings. Although we have difficulties in segmentating and labeling acoustic speech signal, we have explicit rules and theories (syntactic rules and language grammar) to deal with the sequence of symbols. This is actually what a spectrogram reader do: first, he has a highly-developed visual skills to perceive and describe roughly different

speech segments, then he uses his knowledge in phonology to adjust and justify his decisions. What actually might happen is that when we hear an utterance, we cannot get exactly every information from the mere acoustic properties of the speech signal which enters our ears, but also the *"context"* (the signal in front and after) combining with the a priori phonological knowledge (symbol sequence structure) that makes us infer what actually has been said. Of course, there exists even higher level context like the *subject (event)* that is "talking" about.

The use of higher level context is therefore important, if it is not vital. One way to make use of this information is to integrate the constraints imposed by the phoneme context (usually learned from training examples) into the pattern matching module, an typical example is the use of hidden Markov model. Others include separating the two parts and do training individually and unite them together during recognition, like the use of dynamic programming and grammar. We are going to discuss each of them separately.

4.2 HIDDEN MARKOV MODEL APPROACH

In section 2.2, we have seen how hidden Markov model integrates time normalization and pattern matching and trains them together in one step. We are now concentrating on its ability in learning the context information of phoneme (subword units) in hidden Markov model.

Usually in continuous speech recognition, a subword unit is used as the basic

units in hidden Markov model. During training, these subword units are concatenated according to the given lexicon (dictionary) to form different words. In this way, all possibilities of the sequence of the subword units in the training data is tried during training. One point should be noted that there is only one set of parameters, which are the codebook and transition probabilities, associated with each subword units. This is one of the weak point of hidden Markov model where the same set of codebooks is used in different context. The use of context-dependent units (section 3) is therefore introduced, but it needs a retraining of the whole system.

We see that the transitions probabilities between two phonemes is dependent on the frequencies of co-occurrences of that particular two phonemes in any word. In addition, the training data should include all the words in the lexicon. The transition probabilities obtained therefore is highly dependent on the amount of training data, where both the frequencies of co-occurrences of phoneme and the occurrences of words need to be representative enough.

One can actually view the hidden Markov model as a kind of finite state machine with a pre-defined architecture. Each state in the finite state machine is associated with a stochastic output process, which estimate the probabilities of a given speech frame in that state. Therefore the transitional probabilities inferred this way is also dependent on the acoustic properties of the speech vectors and the pre-defined architecture.

4.3 THE DYNAMIC PROGRAMMING APPROACH

Dynamic programming is a widely used tool in operations research for solving sequential decision problem. The first problem in dynamic programming is called an optimal path problem. Consider a set of points labeled from 1 to N. A cost function d(i,j) is associated with point i and point j, which represents the cost of moving from node i to node j. The optimal path problem is then to find a minimum cost function of moving from, say 1 to N as illustrated in Fig. 4.1.

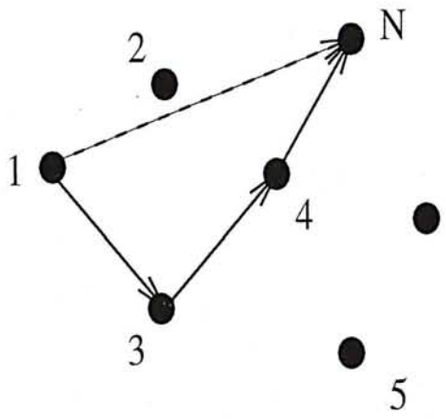


Figure 4.1: The optimal path problem in dynamic programming.

In this case, we have to calculate $\psi_1(1, N)$ and $\psi_2(1, N)$ by

$$\psi_1(1, N) = d(1, N) \tag{4.1}$$

$$\psi_2(1, N) = d(1, 3) + d(3, 4) + d(4, N) \tag{4.2}$$

In continuous speech recognition, dynamic programming was mainly used in the

lexical access stage [31]. Typically, when a sequence of phonemes was obtained from the pattern recognizer (eg. neural network), word matching can be done by searching word by word in the lexicon the most probable word that the phoneme sequence corresponds to (Fig. 4.2). The nearest set of words would then be obtained.

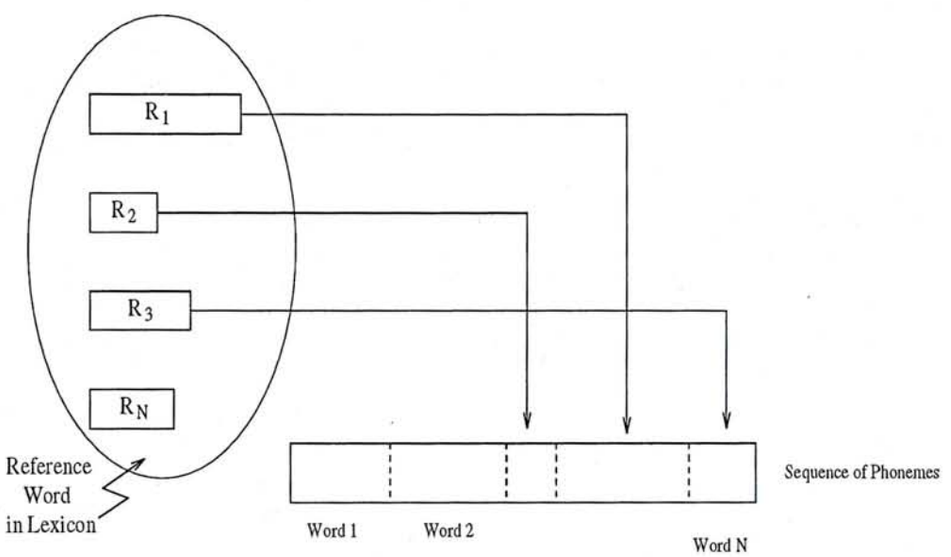


Figure 4.2: The use of dynamic programming method in higher level context.

4.4 THE SYNTACTIC GRAMMAR APPROACH

Syntactic grammars are very often used to capture the structures of patterns in pattern recognition. Given a syntactic grammar, a pattern can be classified as whether it is belonging to this grammar or not. When use in pattern recognition, each class of pattern can be characterized by a grammar, for example, and a given pattern can be classified as belonging to the classes by if it is accepted by the grammar.

In speech recognition, grammars are usually used in the language level, capturing the structures of words. Others have also tried to manually build a grammar to characterize the permissible sequence of phoneme symbols [29]. For example, a grammar can be built from the words *the, that, these and those* as shown in Fig. 4.3.

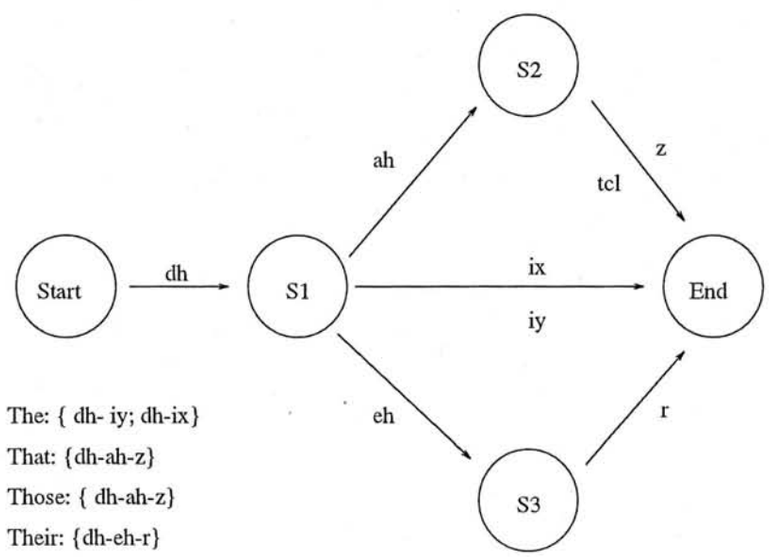


Figure 4.3: The grammar formed by the four words: *The, That, Their* and *Those.*

The grammar inferred from the training data can then be used to search for the word sequence from the tentative phoneme sequence. The detail implementation is to be discussed in chapter 5.

Chapter 5

Finite State Grammar Network

5.1 INTRODUCTION

We have seen briefly how syntactic grammar can be used to deal with the higher level context information in the previous chapter. The advantages and main distinctions from other methods and why it is useful in continuous speech recognition is also shown. Here, the detail description of method is described.

A grammar G can be characterized by a four-tuple,

$$G_c = (V_N, V_T, P, S) \tag{5.1}$$

where V_N, V_T are called the nonterminal symbols and terminal symbols respectively, P is named the set of production rules and S is the set of starting states. Furthermore if G is finite state, then P has the form:

$$P : \alpha \rightarrow \beta \tag{5.2}$$

where

$$\alpha = A; \quad \text{and} \quad \beta = \text{aB or } \beta = \text{a}; \tag{5.3}$$

$$\text{a} \in V_T; \quad \text{A,B} \in V_N \tag{5.4}$$

This chapter describes the detail of the use of finite state grammar (regular grammar) in continuous speech recognition. The set of terminal symbols is taken to be the set of phoneme symbols of the speech data set. In the following sections, the method of learning the syntactic structures of the phonemic transcription of words from examples (grammar inference) is described. Since there are always errors introduced by the pattern classifier, phoneme strings generated are usually noisy. It is very likely that the grammar inferred would reject all the strings of phonemes which directly come from the pattern classifier. Therefore it is necessary to have some measure to detect as well as to correct these errors, which is discussed here also. Some experiments are then carried out to test the performance of the error-correcting grammar inferred, and the results are tabulated here also.

5.2 THE GRAMMAR COMPILATION

5.2.1 Introduction

The simplest way to infer grammar from a set of sentences is to use the canonical finite state grammar G_c [7], where

$$G_c = (V_N, V_T, P, S) \tag{5.5}$$

Now if for any string $x_i \in$ X (training set), let $x_i = a_{i_1} a_{i_2} \ldots a_{i_n}$, we can define the set of production rules as:

$$\begin{array}{ccc} S & \rightarrow & a_{i_1} Z_{i_1} \\ Z_{i_1} & \rightarrow & a_{i_2} Z_{i_2} \\ \vdots & \vdots & \vdots \\ Z_{i_{n-1}} & \rightarrow & a_{i_n} \end{array} \tag{5.6}$$

where each Z_{i_j} represents a new nonterminal symbols. We can denote the set of languages that can be generated from G_c as $L(G_c)$. But it is obvious that there exists many other possible grammar, given a set of training data. If we have a training set I (an inference sample), then the task of grammar inference is to find a grammar G' such that $I \subset L(G')$. For example, if

$$I = \{aab, aaab, aabbb\} \tag{5.7}$$

then one possible grammar is

$$G' = (V_N, V_T, P, S); \tag{5.8}$$

$$V_N = \{A\}; \quad V_T = \{a, b\}; \quad S = \{\sigma\}; \tag{5.9}$$

$$P = \left\{ \begin{array}{ll} S \rightarrow aS, & A \rightarrow bA \\ S \rightarrow aA, & A \rightarrow b \end{array} \right\}; \tag{5.10}$$

where σ is a symbol denoting a starting state. Then,

$$L(G') = \{a^n b^m | n \geq 1, m \geq 1\} \tag{5.11}$$

We see that the grammar G' will have much smaller number of non-terminal symbols and production rules than the canonical grammar generated by the same set of inference set I. Now, consider another grammar G", where

$$G'' = (V_N'', V_T'', P'', S'') \tag{5.12}$$

$$V_N'' = \{A, B, C, D, E\}; \quad V_T'' = \{a, b\}; \quad S'' = \{\sigma\}; \tag{5.13}$$

$$P = \left\{ \begin{array}{ll} S'' \rightarrow aS'', & A \rightarrow bB \\ B \rightarrow aC, & B \rightarrow bD \\ B \rightarrow b, & C \rightarrow bD \\ C \rightarrow b, & D \rightarrow bE \\ E \rightarrow bD, & E \rightarrow b \end{array} \right\}; \tag{5.14}$$

This time the language generated by G" is

$$L(G'') = \{a^n b^{2i+1} | n = 2 \text{ or } 3, i \geq 0\} \tag{5.15}$$

which would be another solution for the same training set I, since $I \subset L(G'')$. We can see that both the number of parameters needed and the *structure* of the symbols sequence captured are quite different in both cases. Depending on the actual applications, one may sometimes want to have the grammar to be more strict in discriminating against symbol sequences that are not in the training set, or sometimes one may want the grammar to relax a little bit. There are a number of ways inferring grammars from the training set samples. One approach is to start from the canonical grammar, where there are various ways

similar non-terminal symbols can be clustered together so as to diminish both the non-terminal symbol and the production rules, without affecting too much the set of language that can be generated from it. We are going to describe one of the popular method, the K-tail clustering method [7] in the next section.

5.2.2 K-Tails Clustering Method

Let $z \in V_T$, and let X denote the set of all possible strings or sentences. Then the *k-tail* of z, denoted by g(z,X,k) is defined as:

$$g(z, X, k) = \{x \in V_T | zx \in \text{X and } |x| \leq k\} \tag{5.16}$$

where $|x|$ denotes the length of string x, and $k > 0$. g(z,X,k) is not defined if there is no string $x \in V_T$ such that $zx \in X$ and $|x| \leq k$. For example, referring back to the example where

$$I = \{aab, aaab, aabbb\} \tag{5.17}$$

then,

$$g(\sigma, I, 3) = \{aab\} \qquad g(a, I, 3) = \{ab, aab\}$$
$$g(\sigma, I, 2) = \emptyset \quad g(a, I, 4) = \{ab, aab, abbb\}$$

The idea of k-tails can be used to define equivalence classes on the states of the cannonical finite state grammar generated by an inference sample I. Let V_1 and V_2 be two distinct states of the canonical finite state grammar, where

$V_1, V_2 \subset V_N$. If

$$V_1 = \{x | z_i x \in I\} \quad V_2 = \{x | z_j x \in I\} \quad z_i, z_j \in V_T$$

then the two states V_1 and V_2 (V_1 and V_2 are called the *tails* of z_i and z_j respectively) are said to be *k-tail-equivalent* if and only if

$$g(z_i, I, k) = g(z_j, I, k) \tag{5.18}$$

For example, let

$$V_1 = \{b, bbb\} \quad V_2 = \{b, ab\}$$

then V_1 and V_2 are said to be 1-tail-equivalent. The use of k-tails equivalence allows us to generate a set of derived grammars based upon the canonical finite state grammar.

5.2.3 Inference of finite state grammar

Another identical but more convenient form to represent finite state grammar instead of production rules is to represent it as *finite state machines*, or *finite state automatas*. A finite state automata is a 5 tuple:

$$A = (X, Q, \delta, q_0, F) \tag{5.19}$$

where X is the set of non-terminal symbols, Q is the set of "states", δ is the transition function, q_0 is the initial states, F is the final states. The transition function δ has its domain: $\delta : Q \times X \rightarrow Q$, which output the set of next state

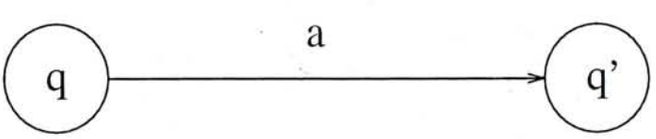
$Q' = \{q'_1, q'_2, \ldots, q'_n\}$ with the current state q and the transition symbol a as input, thus if there is only one possible next state, then

$$\delta(q, a) = q' \tag{5.20}$$

or we can graphical represent the transition function as in Fig 5.1.

It is shown in [6] that given a finite state machine A , there exists a finite state grammar G with

1. $V_N = Q$
2. $V_T = X$
3. $S = q_0$
4. $B \rightarrow aC$ is in P if $\delta(B, a) = \mathrm{C}$; $B, C \in Q, a \in X$
5. $B \rightarrow a$ is in P if $\delta(B, a) = \mathrm{C}$; $C \in F$



δ (q, a) = q'

Figure 5.1: The transition function δ in a finite state automata.

Now given an inference sample I with sequences of symbols, we can define the set of X as all the symbols used in I, Q as the k-tail clustered states, and $\delta(q, z) = \mathrm{q}'$

if and only if $xz \in q'$ and $x \in q$, $z \in X$. The starting states can be represented by σ and $\delta(\sigma, z) =$ q' if and only if z is a starting symbols of any of the sequences in I. The final state can be represented by F and $\delta(q, z) =$ F if and only if there is a string xz in I, where $z \in X$ and q can be reached by x from σ.

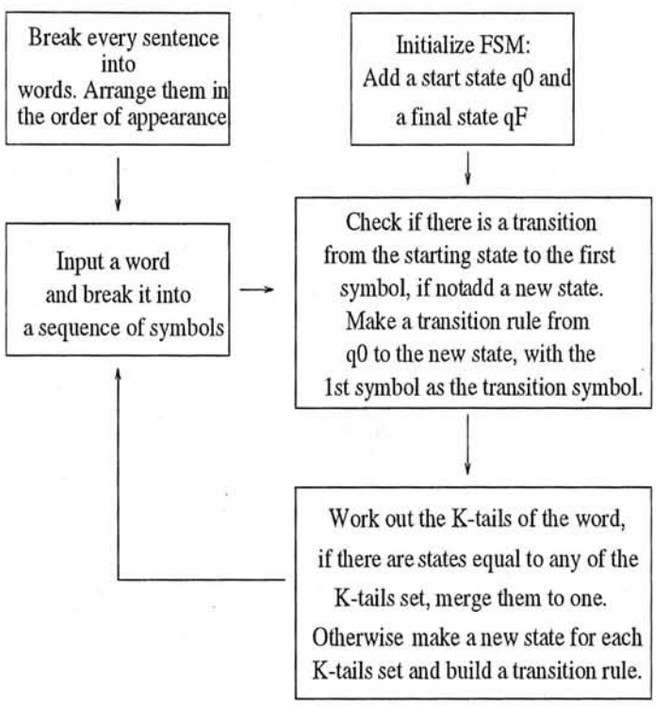


Figure 5.2: The block diagram showing the grammar formation.

5.2.4 Error Correcting Parsing

The grammar G inferred is now capable of deciding whether a given string is acceptable or not by searching the phonemes of the string one by one in the grammar to see if all the transitions are allowed. But owing to the errors introduced by the pattern classifier, phoneme strings generated are usually noisy. Therefore it is very likely that the grammar inferred would reject all the strings of

phonemes which directly come from the pattern classifier. As errors are usually defined in terms of substitutions, insertions and deletions, the matching process can be done in terms of these errors. The procedure for constructing an error correcting parser starts with the modification of a given grammar G by adding the 3 types of error transformations in the form of production rules. In other words, after all the possible next states fail to match the next phoneme, the error correction rules are tried.

If we define L(G) to be the set of sentences (phoneme strings) that can be generated by G, The grammar G is now expanded to G' such that L(G') includes not only L(G), but all possible sentences with the 3 types of errors. Now we define the 3 types of rules of error corrections as follows,

Definition:

1. *Insertion error correction:* $w_1 w_2 \rightarrow w_1 b w_2, for\ some\ \ b \in V_N$

2. *Substitution error correction:* $w_1 a w_2 \rightarrow w_1 b w_2, where\ \ a \neq b, a, b \in V_N$

3. *Deletion error correction:* $w_1 a w_2 \rightarrow w_1 w_2, where\ \ a \in V_N$

where $w_1, w_2 \in V_T$.

During recognition, the phoneme string is parsed and whenever a phoneme cannot be found in any of the next states, the error correction rules are tried. Backtracking is done only when all corrections rule fail. If the end state is reached in the middle of a sentence, a boundary is postulated at that point and is recorded, and a starting state is visited again to seek for the next phoneme.

5.3 EXPERIMENT

The experiment uses all the sentences in the training set of the dialect region 1 of the TIMIT continuous speech database to infer a finite state grammar. In continuous speech recognition, a set of sentences, presented as strings of phonemes, are first broken down into words. It should be noted that these words should include all the vocabularies in the continuous speech recognition system. In order to learn the variations of pronunciations, the nearest phonemic transcriptions are used to represent the actual utterance. Grammar is then inferred from the words, after the parameters of the finite state automata is obtained. The vocabulary comprises 1439 multi-pronunciation words and 61 phonemes. To test the algorithm and the implementation, exhaustive search are carried out for ten sentences where noises are randomly added to the sentences as insertions, deletions and substitution errors, with the total number of errors not more than five. As a result, a total of 490 noisy strings are used. The average number of phonemes in a sentence was about 30. The overall experiment framework is shown in Fig. 5.3.

A noisy phoneme string is first parsed by the error correcting grammar. If the end state is reached in the middle of a sentence, a boundary is postulated at that point and is recorded, and a starting state is visited again to seek for the next phoneme. It is to be shown in the experiment that even if a wrong boundary is detected, the phonemes sequences after that false boundary would rarely have a successful parse all the way to the end of the sentence. Even if it does, the number of corrections needed is large. It is therefore deduced that correct word boundaries entails small number of corrections which is approximately equal to

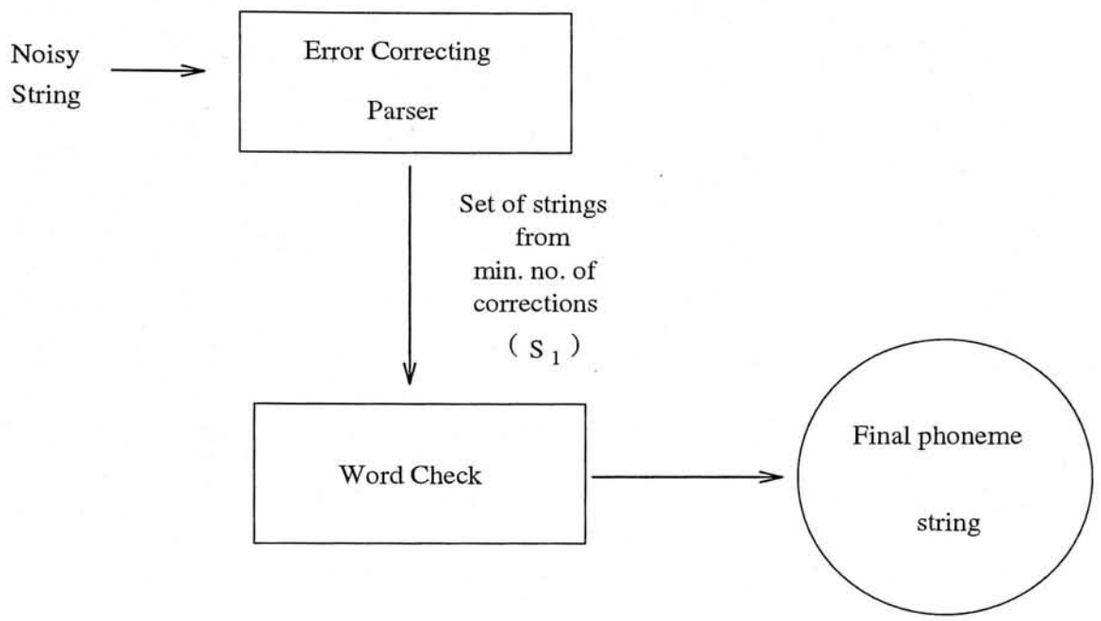


Figure 5.3: Block diagram illustrating the steps of the experiment.

the number of actual errors. This was in fact justified by the experiment. After parsing, a set of phoneme strings (with the corresponding word boundaries) obtained by various numbers of corrections is produced. The minimum number of corrections are then noted and the set of phoneme strings obtained by the minimum number of corrections are then extracted for further processing. It is noted that this set of phoneme strings (S_1) may or may not contain the recovered noiseless string. (In other words, we are not expecting that no noisy phoneme string can be parsed by a fewer number of corrections than the initial errors in the strings.) We then use the word boundaries in S_1 to search for the most probable word sequences by finding the closest words in the dictionary to the words suggested by the word boundaries in S_1. The final phoneme string is then obtained.

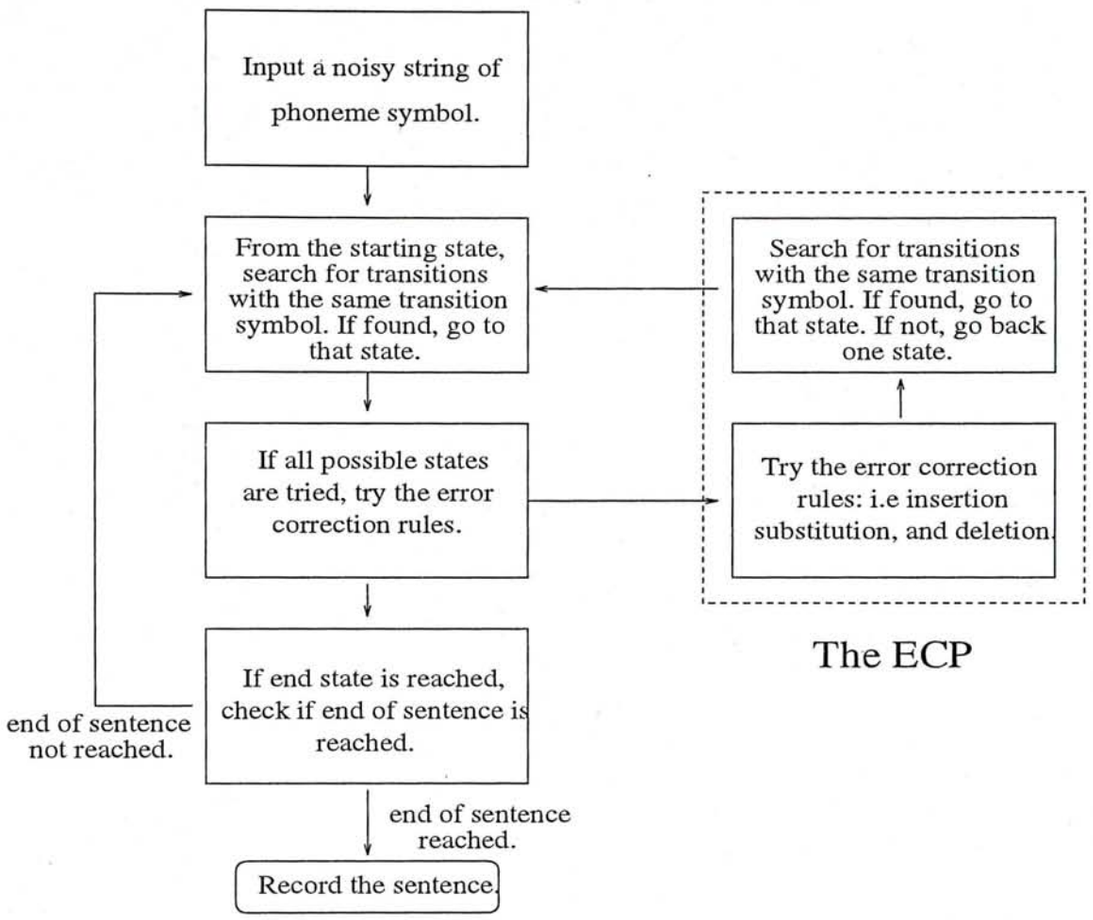


Figure 5.4: Block diagram showing the structure of parser.

One example of the noisy input and the corrected phoneme string with word boundaries is:

Original sentence (corrected sentence) :

The emperor had a mean temper.

$\|dh\ iy\|\ \|q\ eh\ m\ \underline{pcl}\ p\ \underline{r}\ ix\|\ \|hv\ ae\ dx\|\ \|ix\|\ \|m\ \underline{iy}\ n\|\ \|tcl\ t\ eh\ m\ pcl\ p\ axr\|$

Noisy input phoneme string:

dh iy q eh m p ix hv ae **n** *dx ix m* **axr** *n tcl t eh m pcl p axr*

The noisy string shown here has 2 insertion, 1 deletion and 1 substitution errors. All errors are corrected in this case.

5.4 RESULTS AND DISCUSSIONS

The phoneme strings with word boundaries unknown are parsed by the grammar. About 63% (312 out of 490) of the noisy strings have the minimum number of corrections equal to the initial errors, and all errors are corrected in the final phoneme strings. The rest of the strings have only about half of the initial errors corrected and can be classified into two types, namely **Type I and Type II**. Some examples of these two types are shown in Table 5.1, where I5 S0 D0 denotes 5 insertions, 0 substitutions and 0 deletions initial errors in the input noisy phoneme string of the sentence shown previously. The minimum number of corrections needed for the strings to satisfy the grammar are tabulated in the column next to it.

Type I	
Initial Errors	**Min. no. of Corrections**
I5 S0 D0	2
I2 S0 D3	3
I1 S1 D3	3

Type II	
Initial Errors	**Min. no. of Corrections**
I0 S0 D4	3
I2 S0 D2	3
I2 S2 D1	4

Table 5.1: Examples of string with the number of corrections fewer than the initial errors.

Type I in Table 5.1 has the characteristics that the distribution of insertion errors are concentrated on a particular word in the sentence. For example, in

the word *had*, with phonemic transcription *hv ae dx*, if both the *hv, dx* are missing, then it would need only one insertion error correction (deletion of phoneme) to make the whole string a successful parse, fewer than the original 2 deletion errors. Obviously, the problem is more serious if the word is short. (Consider the word "a" which has only one phoneme). Besides insertion errors, we observe from other sentences that other types of errors are also more difficult to correct if the word is short. No words with number of phonemes larger than 5 cannot be recovered in the experiment[1]. We may conclude from this type that word-length is an important factor[2] concerning the recoverability of errors, or in other words, phoneme in a short word is more "significant" than phoneme in a long word.

The second type, shown here as **Type II** , occurs when two words W_A and W_B happens to be close enough in their phonemic transcriptions, typically differing in only one to two phonemes. If the initial errors happens to change W_A to W_B, the parser will yield a successful parse without making any corrections. For example, in Fig. 4.3, the word *those and their* differs only in 2 phoneme. If the *z* in *those* is made to change to *r*, the grammar can change the phonemes *ah* to *eh* or *r* to *z* to yield a successful parse. In that case, although we have a correct word boundaries, the initial errors in the phoneme string cannot be corrected. Worse still, if the word *those* have 2 initial errors where *ah* is changed to *eh* and *r* is changed to *z*, the parser will make no corrections at all in this

[1]Since errors are randomly added, the signal to noise ratio usually increases when the word length increases.

[2]Since the finite state grammar used here are word-based, all context information can only be obtained *inside* the word itself. Therefore the conclusion does not necessarily applies to other grammars which are not word-based, e.g. sentence-based, phrase-based.

word and output the word *their* instead. The finite state grammar in this case obviously cannot recover the errors, which actually cannot be recovered by any other methods (such as HMM) at this level. In order to solve this problem, an understanding level is needed.

One thing can be observed in **Type II** that the ability to correct errors depends on the position of the errors in the word. In [37], the author concludes by calculating the statistics of 96,998 words that spoken English is at least 67% redundant. Obviously, the remaining parts of the word has to be sufficiently informative, distinguishing itself from other words in the dictionary. In **Type II**, if the invariant core of the word is corrupted, the errors introduced cannot be recovered. For example, in the words *those and their*, it is easy to recover the errors introduced to the phoneme *dh*, but difficult to recover the errors elsewhere.

Fig. 5.5 shows the corrections distributions of the sentence: *The Emperor had a bad temper* with 0 insertion errors, 2 substitution errors and 3 deletion initial errors. The vertical axis represents the frequencies of occurrence of the respective type of corrections, while the horizontal axis represents the number of the corrections used to yield a successful parse. Fig. 5.6 shows the corresponding total (Insertion + Substitution + Deletion) number of corrections used. Table 5.2 shows an example of the set of word boundaries obtained by the minimum number of corrections made to the previous sentence. The last one entry in Table 5.2 is the correct word boundaries, shown here as a comparison.

While most continuous speech recognition systems evaluates their performance by word accuracies, most of them are minimizing the classification errors at a

Word boundaries index:
2-10-14-16-20
2-8-12-16-22-29
2-10-12-16-22-29
2-10-14-16-20-28
2-10-14-16-20-28
2-10-14-16-20-28
12-16
3-10-12
2-10-12-16-22
2-10-14-16-20-28 **Correct**

Table 5.2: Word boundaries patterns obtained from the parser of the previous sentence.

sub-unit level. While there exist some essential phonetic features within a word that distinguish itself, the other parts of the word are susceptible to variations. We conclude that the recoverability of phonemes depends very much on the word length and the positions of phonemes. Short words have their phonemes relatively more important than phonemes in long words. And the positions of corrupted phonemes cannot be essential components of the word which distinguishes the word in the whole vocabulary, if it is to be recovered. Therefore it is quite enough to correctly identify these essential features by the pattern recognizer, while the others can be recovered by a postprocessing stage.

On the other hand, word sequence should follow a set of grammatical constraints also, however for simplicity we consider the case where no grammatical constraints are used. The additional constraints in word level should further improve the error-correcting ability of the postprocessor. We believe that there should also exists an "invariant core" part of the sentence, i.e. some words,

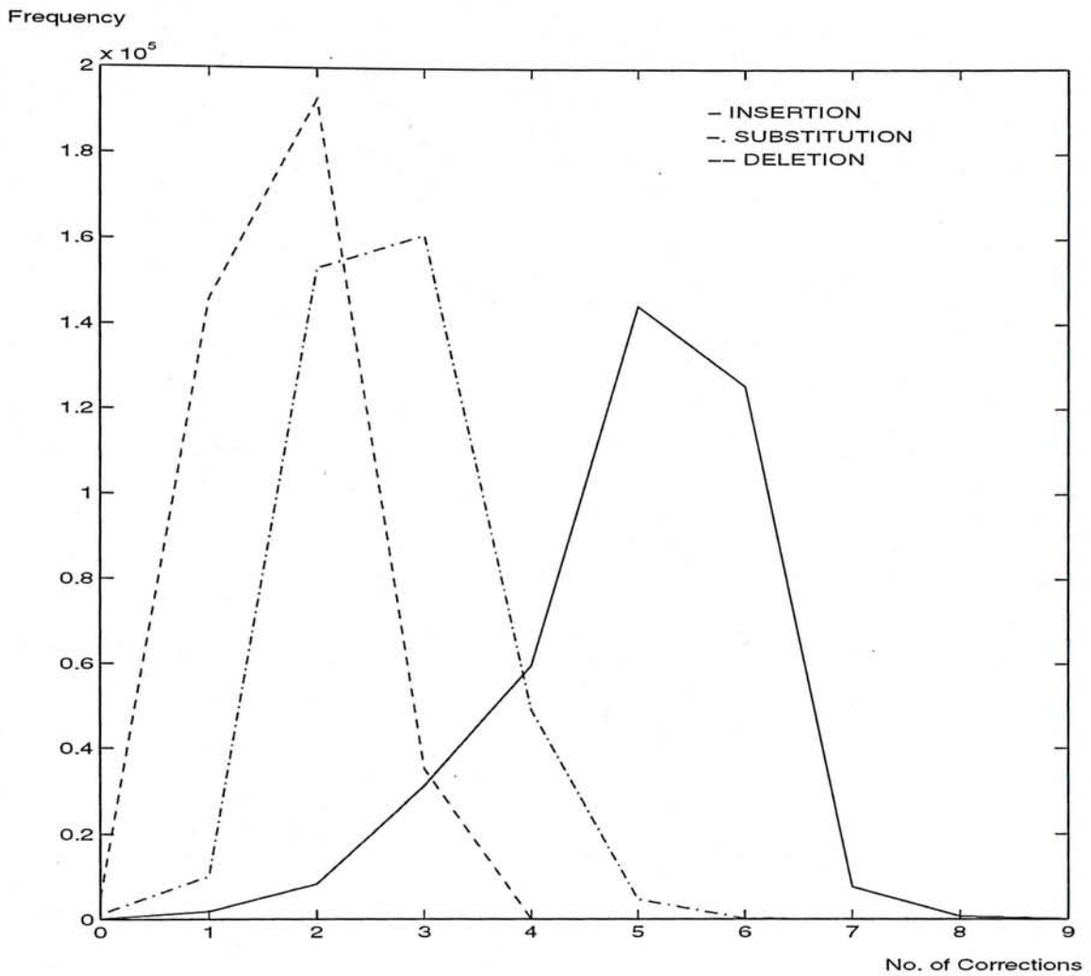


Figure 5.5: Respective corrections distributions during exhaustive search.

which distinguishes the action to be performed defined in the task domain.

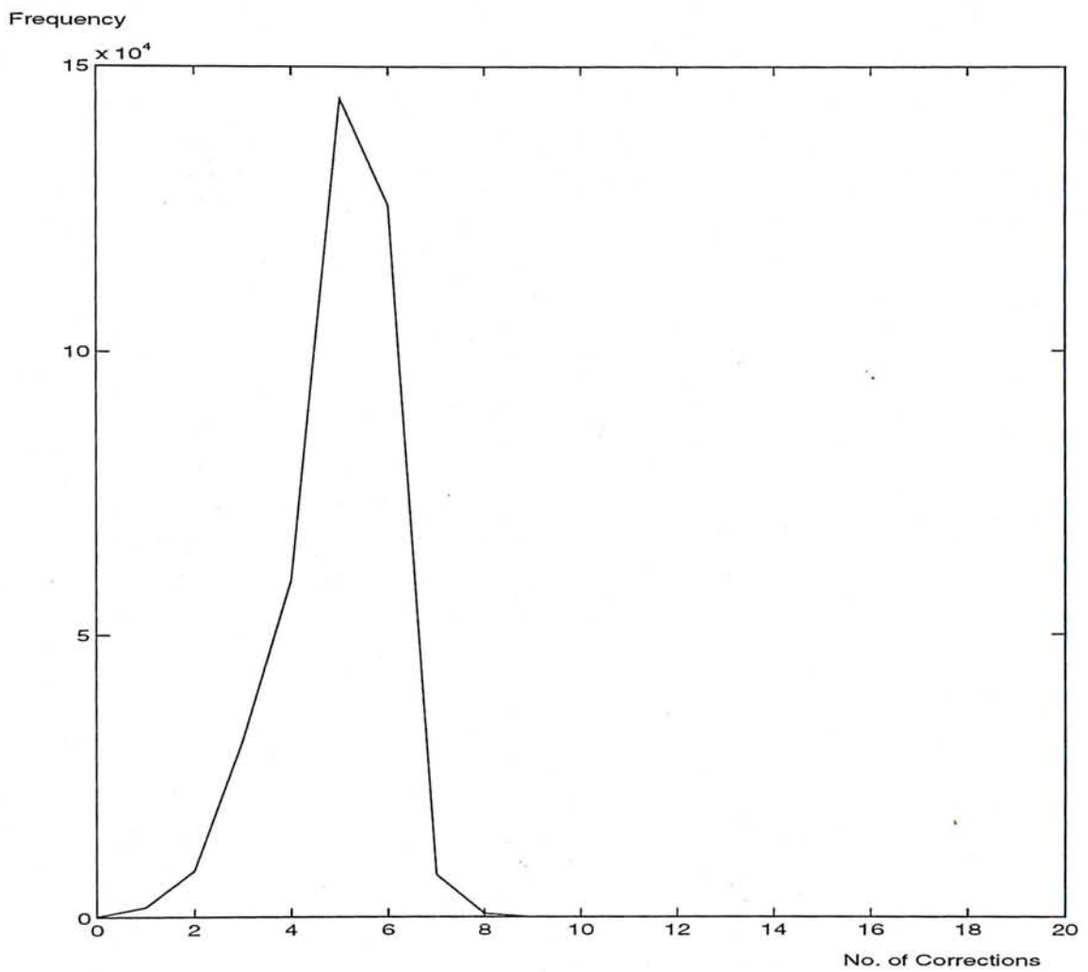


Figure 5.6: Distribution of total number of corrections during exhaustive search.

Chapter 6

The Integrated System

6.1 INTRODUCTION

It is mentioned in chapter 3 that neural network has a good ability in making a mapping of the acoustic vectors to the phoneme symbols. Although this mapping is essential in the whole recognition system, it is not the whole of it. A further mapping of the this sequence of phoneme symbols to the understandable words is also critical. We have already addressed the problem of unclear word boundaries in continuous speech recognition in chapter 2. This problem also determines the performance of the whole system even at the higher level (symbol level) processing. One way to deal with this problem using finite state grammar is described in chapter 5.

An integrated system of neural network and finite state grammar needs a good interface between the two. Specifically, the neural network outputs need to be converted to a linear symbol sequence in order to be compatible with the finite

state grammar, which is called the *postprocessing* of neural network outputs in this chapter. In the following section, the whole integrated system (neural network+finite state grammar) from acoustic vectors to understandable word is described.

6.2 POSTPROCESSING OF NEURAL NETWORK OUTPUT

In chapter 3, it is shown that the neural network output activation can be an indication of how close the acoustic vector is belonging to the particular class that the output node represents. Therefore information about the "identity" of each acoustic vectors is represented by the vector of the activation of the output nodes. In order to measure its performance and to interface to the finite state grammar, a symbolic output is useful, and the whole process of this conversion is described here in this section.

6.2.1 Activation Threshold

It is one of the main problems of continuous speech recognition that an accurate mapping between acoustic vectors and phoneme symbol is difficult to obtain. This problem can be attributed to any of the aspects mentioned in the introduction. Therefore, the winner-take-all strategy usually results to an incorrect classification. The other extreme is to use all the output nodes together with the dynamic time warping algorithm to search for a phoneme symbol sequence, but this might involve a lot of unnecessary calculations, as input pattern which is

totally unrelated to the label of the output nodes typically causes a small value activation. Fig. 6.1 shows the histogram of the activations of three nodes which correspond to the three phoneme symbol: *b,p,aa.* The histogram is obtained from the output nodes of the recurrent neural network with hidden layer trained by the training data set described in the experiment in chapter 3.

It is observed that if the input acoustic vector corresponds to the label on the output node, the neural network output on that node usually gives a large activations with a higher probability. Also, the neural network output nodes corresponding to labels with similar phonetic properties to the correct label gives a larger activation than those which are uncorrelated to it with a higher probability. For example from Fig 6.1, the first row in the figure represents the histogram of the activation at each of the 3 nodes for frames labeled *b*, the second row is for frames labeled *p* and the last row *aa.* The phonemes *b and p* are very similar in their phonetic properties as both are the stops type. We see that it has a larger chance to misclassify *b* as *p* (or vice versa) than to misclassify *b* or *p* as *aa* (*aa* is a vowel which is quite different from the stops). Alternatively, we can say that besides the highest activation output node, other output nodes with *significantly* large activation also give valuable information about the identity of the input acoustic vector. This leads to the use of multiple output nodes as the candidates for the finite state grammar.

Table 6.1 gives the percentage of correctness with different activation thresholds. The percentage of correctness is calculated by first picking out those output nodes with activation higher than the activation threshold. If those phonemes picked out contains the correct label, a correct classification is counted at that

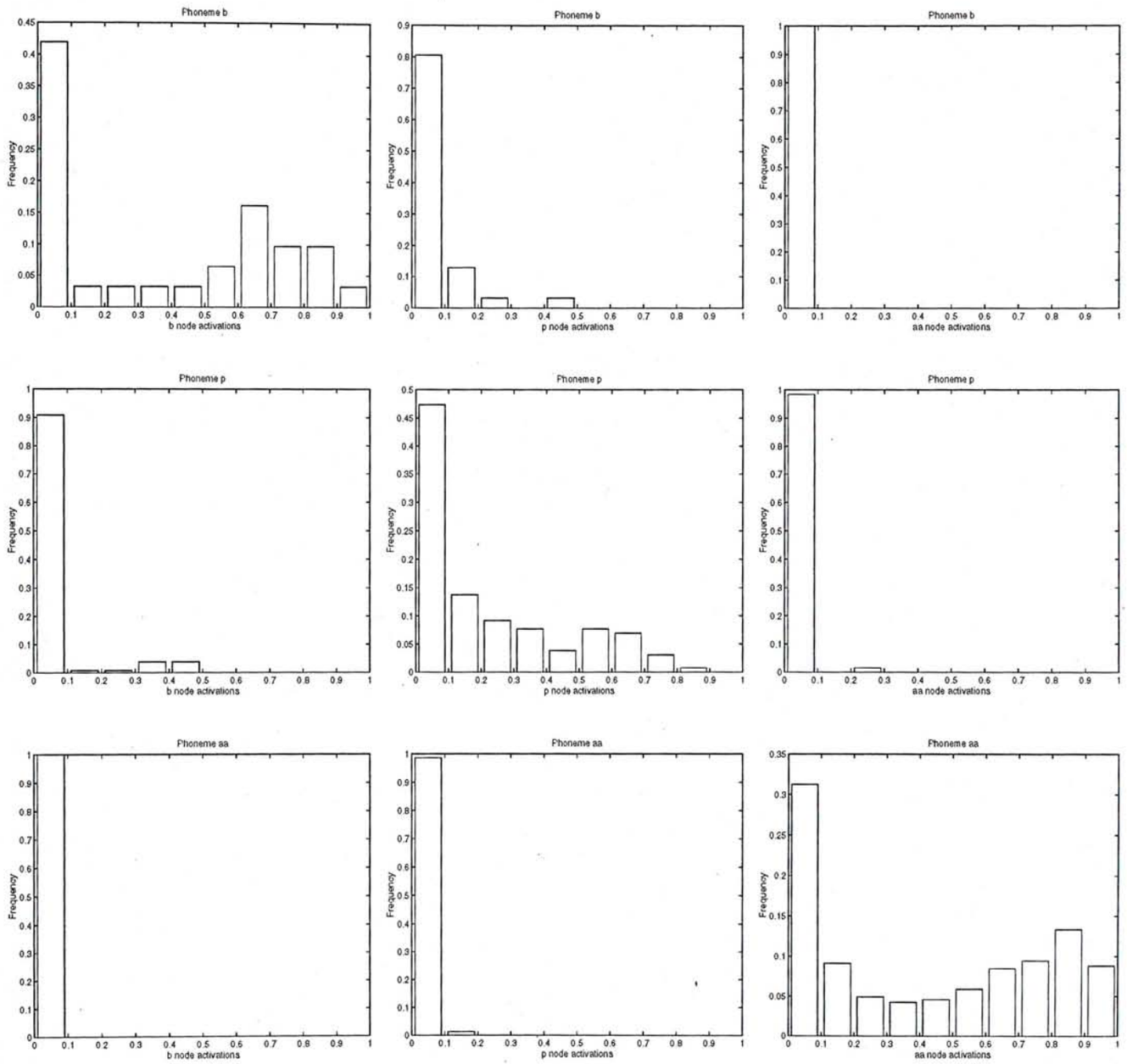


Figure 6.1: Histograms of activations of neural network output nodes of the phonemes *b*, *p* and *aa*.

Activation Threshold	% Accuracy	Ratio
0.1	45%	2.7
0.2	39%	1.8
0.3	26%	0.8
0.4	24%	0.6
0.5	22%	0.5

Table 6.1: Table showing relationship between activation threshold value and accuracies.

frame. Obviously, a lower activation value gives a much higher correctness, in the expense of more alternative phoneme symbol candidates. We can see the relationship between the two in Table 6.1. The column *ratio* indicates the average number of picked out nodes per frame. It is obtained by first counting the number of output nodes with activation higher than the given threshold, then divided by the total number of input frames.

Since a low activation threshold will pick out a lot of candidates for the finite state grammar (increasing insertion errors[1]), and a high activation threshold will lower the chance of picking out the correct phonemes, a compromise has to be made when choosing the threshold. Here an activation threshold of 0.2 is chosen.

6.2.2 Duration Threshold

Since each phoneme symbol usually consists of several consecutive frames in each word, a merging process is needed to find out the *duration* of the phoneme.

[1]It is possible that not only the number of candidates but also the number of *time slots* are increased. Therefore it is likely to have spurious phonemes that cannot be recovered by the finite state grammar.

It is found that consecutive frames may or may not be consistently giving out high activations at a particular phoneme symbol. For example, if 5 consecutive frames of speech should belong to phoneme P_1, the actual output activation after thresholding might becomes something like:

Phonemes picked out	P_1 P_2	P_2 P_4	P_1 P_5	P_3 P_6	P_1 P_7
Frame no.	F_1	F_2	F_3	F_4	F_5

where each column represents the picked output phoneme symbols in each of the 5 consecutive frames F_i. In this example, 2 phonemes are picked out at each frame, though the number of phonemes picked out in each frame may not be the same in actual situation.

In order to deal with this problem, some constraints should be imposed in order to avoid assigning erroneous phoneme labels to *frames*. One straight forward method is to make use of the mean duration of the phonemes. If the number of picked out phoneme label in consecutive frames is too small or too large relative to the mean duration of the phoneme, the probability of the phoneme being the correct labels in these frames is small. In the above example, if the mean duration of P_2 is 10, then it is likely that P_2 is not the correct labels to F_1 and F_2.

Specifically, the procedures are as follows. First, the number of occurrence N of a particular phoneme within a window is counted. The length of this window is equal to the mean duration of that phoneme, which is calculated from the training data set. In the above example, let the mean duration of P_1, denoted

as $Mean(P_1)$, be calculated from the training set to be 5, and the number of occurrence of P_1, denoted by $N(P_1)$, is 3. Then, some threshold value has to be chosen such that the number of occurrences of the phoneme is "reasonable" by comparing it to the mean of the phoneme. Now if we denote the *duration threshold* to be D_{thres} and

$$\frac{N(P_i)}{Mean(P_i)} > D_{thres} \tag{6.1}$$

then the number of occurrences of the phoneme is considered "sufficiently large" and the label P_i can be assigned to the frames inside the window. Since it is possible that the duration of phoneme P_i is larger than its mean duration, if condition 6.1 is satisfied, the window is shifted one frame to the right (window length unchanged) and $N(P_1)$ is re-counted. Condition 6.1 is tested again, and if it is satisfied, possible duration of P_i is incremented. If condition 6.1 is not satisfied, the duration is not incremented but is reckoned as the duration of the phoneme P_i. The same process repeats until it violates condition 6.1 or the total duration of the phoneme reaches a upper limit D_U which is chosen to be:

$$D_U(P_i) = Mean(P_i) + 3 \times STD(P_i) \tag{6.2}$$

where $STD(P_i)$ is the standard deviation of phoneme P_i calculated from the training set. The duration upper limit is set in order to avoid impossible duration of phonemes. A three standard deviations is chosen in this case.

A duration lower limit D_L is also set and defined as:

$$D_L(P_i) = max \{ Mean(P_i) - 3 \times STD(P_i) , 0 \} \quad (6.3)$$

so that if the duration of a particular phoneme is smaller than D_L, it is discarded. This is to ensure that erroneous phonemes would not be selected for further processing. Three standard deviations is also chosen here to do the discrimination.

After these steps in setting the duration threshold, a table of output phoneme symbols with their corresponding starting and ending times (frames) is obtained. This is then used for further processing in the later sections.

6.2.3 Merging of Phoneme boundaries

After the previous step, a table of possible phonemes with their possible starting and ending frame numbers are obtained, which usually has many overlappings. In order to obtain a linear representation of the neural network outputs, some criteria have to be set in order to obtain a clear-cut phoneme boundaries. If P_1 and P_2 are two phoneme intervals obtained and P_i^s and P_i^e are the starting and ending times respectively of phoneme P_i, then there are 3 possible cases for the relationships of P_1 and P_2:

1. $P_1^s < P_2^s$ and $P_1^e > P_2^e$. (Fig. 6.2) Then P_1 and P_2 are assumed to be in the same time slots, or in other words, they are merged together.

2. No intersection. (Fig. 6.3) If $P_2^s = P_1^e + \Delta t$, and $\Delta t < \Delta_{sil}$, where Δ_{sil} is a pre-defined threshold and two consecutive different time slots are assigned to P_1 and P_2. If $\Delta t > \Delta_{sil}$, a phoneme is assumed to be misclassified as

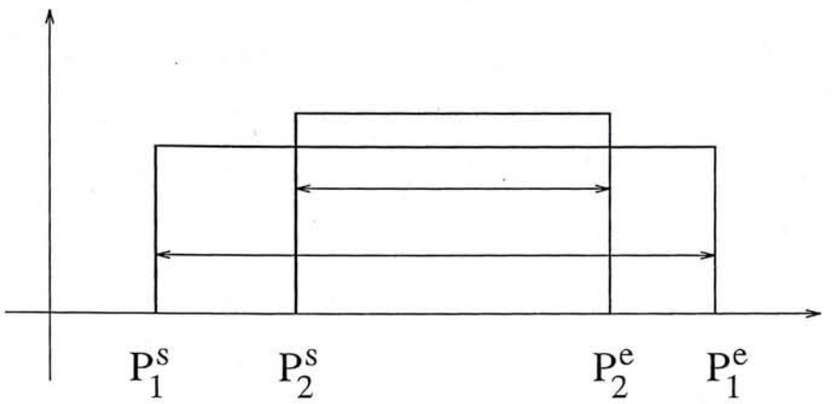


Figure 6.2: $P_1^s < P_2^s$ and $P_1^e > P_2^e$.

silence from P_1^e to P_2^s, the number of frames $\Delta_{sil} = P_2^s - P_1^e$ is recorded as a time slot between P_1 and P_2. Here Δ_{sil} is chosen to be 5.

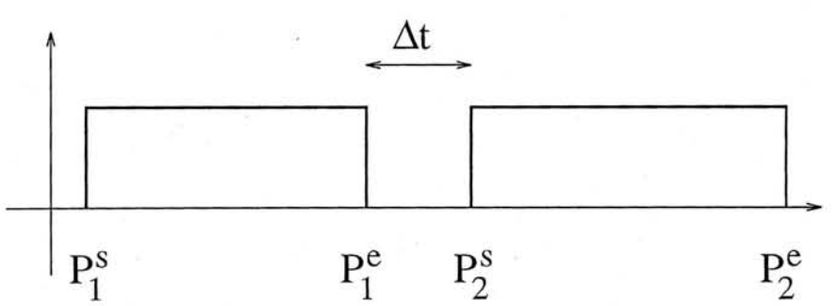


Figure 6.3: No intersection.

3. P_1 and P_2 partially intersect. (Fig. 6.4) If we assume $P_1^e > P_2^s$, $P_2^e > P_1^e$ and $P_1^e - P_2^s = \Delta t$, then if $\Delta t <$ certain threshold ΔT_{thres}, P_1 and P_2 are assigned consecutive time slots, otherwise, it is merged to the same time slot. Here ΔT_{thres} is chosen to be 2.

The distributed output representation is changed to a sequence of linear time slots within which possible phonemes are picked out. This sequence of phoneme

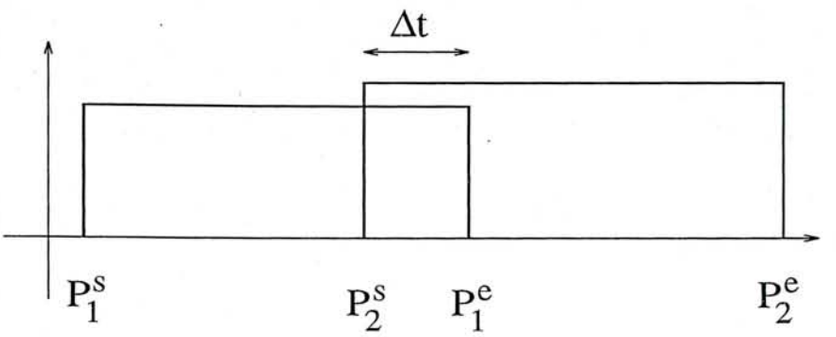


Figure 6.4: Partially overlap.

string is then ready to be parsed by the finite state grammar.

6.3 THE ERROR CORRECTING PARSER

Basically, the parsing algorithm of the phoneme string is the same as the original finite state grammar described in chapter 5. The only change made is to the error correction rules defined in section 5.2.4 in page 69, which are changed in order to meet the characteristics of the phoneme strings from the neural network outputs. The criterion for the 3 types of error corrections are now:

1. **Deletion error[2] correction**

   Since the database is designed so that there is no pauses between words[3], all frames classified as silence (i.e. the Δ_{sil} defined in the previous section) in the middle of the sentence must be incorrect. Phonemes are then added to these frames to "fill in" the time slots. As it is possible to have more than one phoneme "missing", the insertion of phoneme in this time

[2]Deletion error is the error introduced by deletion of phoneme, which needs *insertion* of phoneme to correct.

[3]The database is designed where no frame in the middle of the sentence is labeled silence.

slot is continued until the sum of means of the inserted phonemes exceeds the number of frames which are misclassified as silence (Δ_{sil}). Also, by selecting a small enough ΔT in the merging steps in the postprocessing stage of the recurrent neural network, deletion error correction is carried out only at this kind of "silenced" time slots.

2. **Insertion error[4] correction**

   Insertion errors usually occur in time slots containing short-durationphonemes. As there is a duration lower limit in the postprocessing stage of the recurrent neural network and by choosing a suitable duration lower limit, it is not possible to have two consecutive time slots containing phonemes that are spurious, or in other words, there should always be "some phonemes" deletion of consecutive time slots is prohibited. Also, a threshold for deletion of "silenced" frame is also set to avoid discarding a large number of frames.

3. **Substitution error correction**

   It is observed that the misclassified phonemes by the recurrent neural network are still information carrying. This is shown in Fig. 6.1 in section 6.2 in page 85. That is to say, the misclassified phonemes are usually acoustically similar to the correct phoneme (as the *b and p*) in Fig. 6.1. Classes can be chosen among the set of phoneme which are close enough in their physical features, and a hierarchy of similar classes can be built. According to [27], the original 61 phoneme symbols used in the TIMIT database

[4]Likewise, insertion error correction needs deletion of phoneme to correct.

can be classified into different classes according to their acoustic similarity, as shown in Table 6.2. According to these classes, a hierarchy of the acoustic similarity [27] between these classes can be grouped, where the higher the level in the hierarchy, the more "similar" the phonemes within each element of the groupings are. The hierarchical grouping chosen are shown in Table 6.3.

Grouping 0	
Voiced stop	b d g dx q pau epi
Voiceless stop	p t k dx q pau epi
Voiceless sibilant fricative	s sh ch
Voiced sibilant fricative	z zh jh
Voiceless non-sibilant fricative	f th
Voiced non-sibilant fricative	v dh
Nasal	m n ng em en eng
Liquid	l r el
Glide	w y hh hv
Front vowel	iy ih eh ey ae
Middle vowel	aa ah ao er ax ix axr ax-h
Back vowel	ow uh uw ux
Diphthong	ey aw ay oy
Closed voiced stop	pau epi bcl dcl gcl
Closed voiceless stop	pau epi pcl tck kcl tcl

Table 6.2: Table of the classes of phonemes

Now substitutions corrections can be weighted according to the hierarchy of classes. The method is to add one more correction whenever it needs to descend one level in the hierarchy to find the original phoneme belonging to the same class of the new phoneme. This is done in order to discriminate the substitutions of phoneme which are "far" away from the original neural

network "guess". Whenever substitution is carried out, phonemes within the same class of the classified phoneme defined in Table 6.2 is tried first. If all of them fail, the phonemes in the same class defined in Group 1 is tried, but if it success, 2 substitution corrections are attributed to this correction. The same applies to Group 2 and Group 3, where 3 and 4 substitution corrections is recorded for that phoneme substitution. Thus the error correcting parser is penalizing "far" away phoneme substitutions, while it cuts down a lot of search paths in the grammar at the same time.

Grouping 1	Stop
	Voiceless fricative
	Voiced fricative
	Liquid and Glide
	Front and Middle vowel
	Middle and Back vowel
	Closed voiced and voiceless stop
Grouping 2	Stop and Closed stop
	Voiceless fricative and Closed stop
	Nasal, Liquid and Glide
	Liquid, Glide Front, Middle, Back vowel
Grouping 3	Voiced and Vowel
	Voiceless

Table 6.3: Table of the hierarchy of classes of phonemes

It is to be noted that the error correction strategy is now looking for weighted distance (defined in terms of the corrections) phoneme strings in the search space defined by the grammar. It depends very much on the ability of the recurrent neural network to classify "close enough" to the desired class. Consider

the phoneme string of the word *abruptly* obtained from the recurrent neural network output:

uh	bcl	dh	r	ih	-6	b	n	ih
ih	ax	k	ix			k	l	iy
F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9

For unweighted correction, the one of the outputs would be:

dh	ax	dh	ix	q	ey	n	ih
a		the		a		near	

which takes only 3 substitutions and 1 insertion error correction. But the substitutions it takes are:

F_1	uh or ih	$\rightarrow$	dh
F_5	ih	$\rightarrow$	q
F_7	b or k	$\rightarrow$	ey

From the misclassified phoneme output of the neural network, it is very probable that F_1 and F_5 are vowels and F_7 is a stop. The substitution made by unweighted correction is therefore ignoring the information given by the recurrent neural network output. The use of weighted error correction is based on the assumption that even if the recurrent neural network misclassified a frame of speech, it should not be very far away from it. In the above example, the output with weighted substitutions would be:

ah	bcl	b	r	ah	pcl	p	l	iy
abruptly								

and the substitutions it takes are:

F_1	uh or ih	$\rightarrow$	ah
F_3	dh or k	$\rightarrow$	b
F_5	ih	$\rightarrow$	ah
F_7	b or k	$\rightarrow$	p

According to the groupings in Table 6.3, the number of substitutions made in the two cases are 15 (5 + 5 + 5) and 7 (2 + 2 + 2 + 1) for the weighted and 3 and 4 for the unweighted substitutions respectively.

One reason to justify the use of the weighted error correction is that, without the weights in substitution, we are actually guessing the correct phoneme by its mere context, treating the output of the recurrent neural network in a "all-or-nothing" principle. We know that wild guess of phoneme by context suffers from the disadvantage of dependencies of the phoneme on its position in and length of the word. While the recurrent neural network will misclassify at any position within a word and occurs in words of any length, the correction of misclassified phoneme on both the position and word length are bound to be causing a failure. The weight measure makes better use of the information from the recurrent neural network to recover the above weak points.

6.4 RESULTS AND DISCUSSIONS

The experiment is carried out by combining the recurrent neural network and the finite state grammar. The testing dataset is used to feed in the trained recurrent neural network and the outputs obtained are postprocessed as described in section 6.2. The whole process can be represented by the block diagram in Fig. 6.5.

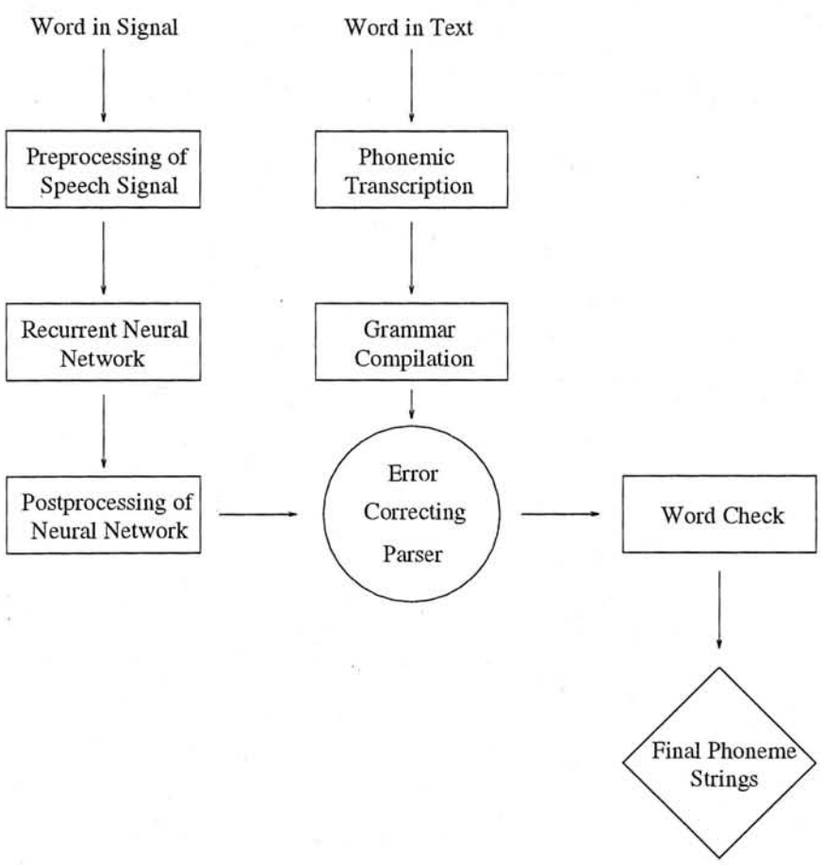


Figure 6.5: Block diagram of the integrated system.

The error correcting parser is weighted according to the hierarchy in Tab 6.2 and Tab 6.3. The input phoneme string after postprocessed from the neural network

is parsed by the grammar. Whenever a state is reached where no possible forward paths matches the next phoneme, the error correction rules are tried. If all the error correction rules fail, the previous state is traced and alternative paths other than those leading to the state just visited is tried (*backtracking*). Whenever a successful parse is done, the number of corrections made is recorded, and the minimum number of corrections is updated also. If in the middle of the parse, the number of corrections used exceeds the minimum number of corrections by two[5], backtracking is done immediately. This cuts down the amount of searching a lot. The output possible phoneme strings with word boundaries are checked against the dictionary. The remaining phoneme strings forms the final set of phoneme strings.

The first experiment using only weighted substitutions results in a large number of sentences in the final phoneme string set. Tab 6.5 shows the number of phoneme strings in the set of final strings of each of the sentences. We see that not only there is a large number of sentences, but also there is a large variations in the number among the sentences. This can be attributed to the existence of a relatively large duration of "silenced" frame (the Δ_{sil} in section 6.2.3 in page 88). Typically, when the number of "silenced" frames can allow more than 5 phonemes to be inserted, a large number of combinations exists. (The insertion of multiple phonemes depends only on the phonemes before and after the insertion.) The result is that several short words (words with less than 3 phonemes) are inserted into this intervals, so long as these short words can "link" together the 2 ends of the "silence" frames. Therefore, a weighted insertion is

[5]Here, phoneme strings with the minimum number of corrections and minimum number of corrections plus one is aimed at, which is an adjustable parameter.

introduced in the second second experiment, where 2 classes are chosen this time as shown in Tab 6.4.

Group 1	Stops and closed stops	b d g dx q pau epi p t k dx q bcl dcl gcl pcl tck kcl tcl
Group 2	The rest of the phonemes in Tab 6.2	s sh ch z zh jh f th v dh m n ng em en eng l r el w y hh hv iy ih eh ey ae aa ah ao er ax ix axr ax-h ow uh uw ux ey aw ay oy

Table 6.4: The grouping for weighted insertion.

A division of phonemes into more classes for weighted insertions have been tried, but no significant changes were observed. The number of sentences obtained are also shown in Tab 6.5.

	Weighted Substitutions only	Weighted Substitutions and Insertions		Weighted Substitutions only	Weighted Substitutions and Insertions
sx1	30	2	sx9	10	10
sx2	29	2	sx10	237	11
sx3	309	2	sx11	2835	2
sx4	22	2	sx12	20	3
sx5	16	4	sx13	2	1
sx6	92	2	sx14	8	2
sx7	43	5	sx15	3	2
sx8	10	1			

Table 6.5: The number of sentences generated out from the parser.

We can see the dramatic effect on the number of sentences in the final phoneme string set of weighted error corrections in Tab 6.5. The big difference in the number of sentences between the two can be explained by that "wild guesses" from context (phoneme symbol in front and after) has a lot of alternatives compared to weighted error corrections. In fact, another way of doing this is to build a confusion matrix and use those entries to guide the error corrections, which may further cut down the searching steps, but no attempt was made to achieve this in this thesis.

Tab 6.6 shows the result of the *phoneme* recognition with weighted insertions and substitutions. Accuracies is calculated by 100% - % of insertions - % of deletions - % of substitutions errors. % of Correctness is calculated by 100% - % of deletions - % of substitutions errors.

Correctness	Insertions	Deletions	Substitutions	Accuracy
69.67%	1.46%	7.32%	23.01%	68.21%

Table 6.6: Results of phoneme recognition of the integrated system.

Tab 6.7 shows the result of the *word* recognition weighted insertions and substitutions. The results are tabulated by using the word boundaries obtained. Since a word check has already been done, all words in the final phoneme strings are already "legal". It should be noted that there are more than one output string from the parser. For a practical continuous speech recognition system, some measures (word grammar and semantics) should be included to yield the most probable output sentence. Here, no grammar has been imposed in the word

level.

Correctness	Insertions	Deletions	Substitutions	Accuracy
53.9%	24.05%	5.6%	40.5%	29.85%

Table 6.7: Results of word recognition of the integrated system.

A comparison of the results to the existing systems are made in the chapter 7 conclusions.

Chapter 7

Conclusions

This thesis concentrates on the use of context information in continuous speech recognition. It was shown that the common steps in the whole speech recognition process can be broken down into two levels: the acoustic (low) level, and the symbol (high) level. Besides, brief review of the use of context information in labeling speech frames are made in both levels. It is concluded that context information is essential as both the acoustic signal and the labeling process are context-dependent. A lot of effort has been made during the last decades in capturing between-frames interdependence, in the expense of heavy computations. It is known that in order to have a good coverage of the contextual variations of phonetic features in phonemes, a huge amount of training data is needed. In the experiment, an obviously insufficient training data set was used in the acoustic level and not surprisingly, the results of the pattern recognizer is not comparable to the existing systems.

Besides the acoustic level, this thesis has shown an alternative way which deals

with this problem in the symbolic (word) level. Even though there are not enough training data to capture the phonetic contextual variations, the use of higher level context information can achieve comparable results of most of the continuous speech recognition system. Since it needs huge amount of training data to capture every possible contextual variations of phonemes in the acoustic level, heavy computational resources are required. It was shown that the use of high level context information can obtain comparable results to the existing systems with much less requirements of computations and memories.

This thesis has shown the use of recurrent neural network and finite state grammar in continuous speech recognition. The results presented in section 6.4 compare favourably with other TIMIT results. For phoneme recognition, Robinson [29] reported a 63% classification rate using the full TIMIT dataset ("si" and "sx" sentences) with 840 sentences for testing. The frame-by-frame accuracy is reported to be at maximum 66% while here only a 35% maximum is reported. This maybe attributed to the fact that only a small training dataset[1] is used and the generalization power is therefore comparatively weak. This problem can be overcomed with sufficient computational resources, where the whole training set can be used to provide a better coverability of the speech data.

For continuous speech recognition system using the TIMIT dataset with techniques other than neural network, Lee & Hon [23] reported a 73.8% correctness (66.1% accuracy) with a 39 symbol set using hidden Markov model with multiple codebooks. Kapadia [17] reported a 74.4% correctness (69.3% accuracy)

[1] As the perplexity of the TIMIT dataset is as high as 2000, the training dataset is undoubtedly insufficient

using maximum mutual information (MMI) training of hidden Markov model. As the TIMIT database is basically designed for phoneme recognition, not too much word recognition results are available for comparison. In [2], a 25.2 % word accuracy is reported and a 28% word accuracy is obtained when 9 frames are used at the input of the multilayer perceptrons in a hybrid system combining hidden Markov model and neural network.

When comparing hidden Markov model and neural network, neural network shows a better capability to incorporate multiple constraints and finding optimal constraints for classification [2] and better discriminant ability than the Gaussian classifier in hidden Markov model, but it suffers from its poor ability to capture temporal sequences. While the combination of neural network and hidden Markov model is showing encouraging results, the strong assumption of hidden Markov model about the statistical distributions of input features is still a main problem. Also, hidden Markov model suffers from the major weakness that its architecture (states) are pre-defined and the obtained optimization process is converging to a set of maximum likelihood, which is poor in discriminative ability[2].

The use of recurrent neural network focuses on the poor ability of multilayer perceptrons in handling temporal sequence classification. The addition of feedback in multilayer perceptrons incorporates long-term context efficiently as an infinite impulse response filter. It is possible that uncertain underlying long-term contextual information is accumulating over speech frames which, if captured

[2]The MMI approach [17] provides more discrimination but the mathematics become more complicated, and many assumptions have to be made also.

properly, can ease the classification of phonemes. The results in section 6.4 shows a good comparison of static neural network and recurrent neural network in continuous speech recognition. Of course, if these long-term contextual variables are to be captured properly, a larger training set with better coverage has to be used, which in turns, required larger computational power. One major problem with neural network is its training time. For most of the experiment in section 3.3, it takes several weeks to finish the training. Therefore future work should be directed to improving the efficiency of the training algorithm.

In [29], it is shown that the use of bigram probabilities (the probabilities of co-occurrences of every 2 phonemes) improves the performance from 36.8% to 63.3% in accuracy in the experiment of phoneme recognition using neural network. In [37], it was shown that spoken English is at least 67% redundant, which implies that there are a lot of dependencies among phonemes in every phoneme sequence. Since syntactic grammars (structural pattern recognition approach) are good at capturing the structure of patterns, it is used to capture these dependencies. It is shown in section 6.4 that even if the performance in local classification (frame) is poor, finite state grammar can correct many of the errors made by the neural network, yielding a good global (symbol) classification performance. Possible future work includes a better clustering technique in the compilation of grammar, which should be more specifically suited to the characteristics of the phoneme sequences. For extension to large vocabulary and truly speaker-independent continuous speech recognition system, word and semantic level process should be included either as a integrated 3-level paring or as separate steps.

Bibliography

[1] G. BALL and D. Hall. "A clustering technique for summarizing multivariate data". *Behav. Sci*, 12:153 – 155, 1967.

[2] H. Bourlard and N. Morgan. "Continuous speech recognition using multilayer perceptrons with hidden Markov models". In *Proc. ICASSP*, 1990.

[3] L. Devillers C. Dugast and X. Aubert. "Combining TDNN and HMM in a Hybrid System for Improved Continuous-Speech Recognition". *IEEE Trans. on Speech and Audio Processing*, 2:217–223, 1994.

[4] P.L. Cerf, W. Ma, and D.V. Compernolle. "Multilayer Perceptrons as labelers for Hidden Markov Models". *IEEE Trans. on Speech and Audio Processing*, 2:185 – 193, 1994.

[5] J.L. Flanagan. *"Speech analysis, synthesis and perception"*. Springer-Verlag, 1972.

[6] K.S. Fu. *"Syntactic Pattern Recognition, Applications"*. Springer-Verlag Berlin Heidelberg, New York, 1977.

[7] K.S. Fu. *"Syntactic Pattern Recognition and Applications"*. Prentice Hall, Englewood Cliffs, New York, 1982.

[8] S. Furui. "Speaker Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum". *IEEE Trans. ASSP*, pages 52 – 59, 1986.

[9] P.D. Green, G.J. Brown, M.P. Cooke, M.D. Crawford, and A.J.H. Simons. "Bridging the gap between signals and symbols in speech recognition". In *Advances in speech, hearing and language processing*, volume 14. JAI Press, London, 1990.

[10] J. Harrington and G.W. Cooper. "Word boundary detection in broad class and phoneme strings". *Computer Speech and Language*, pages 333–358, 1993.

[11] X.D. Huang and M.A. Jack. "Semi-continuous hidden Markov models for speech signals". *Computer speech and language*, 1989.

[12] K. Ishizaka and J.L. Flanagan. "Synthesis of Voiced sounds from a two-mass model of the vocal cords". *Bell System Tech. J*, 1972.

[13] H. Iwamida, S. Katagiri, and E. McDermott. "Speaker-independent large vocabulary word recognition using an LVQ/HMM hybrid algorithm". In *Proc. ICASSP*, pages 497–500, 1991.

[14] F. Jelinek. "The development of an Experiment Discrete Dictation Recognizer". In *Proc. IEEE*, 1987.

[15] B.H. Juang, L.R. Rabiner, and J.G. Wilpon. "On the use of bandpass filering in speech recognition". *IEEE Trans. on ASSP*, pages 947 – 954, 1987.

[16] B.L. Kalman and S.C. Kwasny. "TRAINREC: A system for training feed-forward and simple recurrent networks efficiently and correctly". *Technical report, Dept. of computer science, Washington University*, 1993.

[17] S. Kapadia, D. Valtchev, and S.J. Young. "MMI training for continuous phoneme recognition on the TIMIT dataabase". In *Proc. ICASSP*, 1993.

[18] T. Kohonen. *"Self-organizing and Associative memory"*. Springer-Verlag, Berlin, 1984.

[19] M. Kurimo and K. Torkkola. "Application of Self-Organizing Maps and LVQ in training continuous density hidden Markov models for phonemes". In *International Conference on Spoken Language Processing*, volume 1, pages 543 – 546, 1992.

[20] C.H. Lee, L.R. Rabiner, R. Pieraccini, and J.G. Wilpon. "A segment model based approach to speech recognition". In *Proc. ICASSP, New York*, pages 501–504, 1989.

[21] C.H. Lee, L.R. Rabiner, R. Pieraccini, and J.G. Wilpon. "Acoustic Modeling for large vocabulary speech recognition". *Computer Speech and Language*, pages 137–165, 1990.

[22] K.F. Lee. *"Automatic speech recognition - the development of the SPHNIX system"*. Kluwer acadamic publishers, Boston, 1989.

[23] K.F. Lee and H.W. Hon. "Speaker-Independent phone recognition using hidden Markov models". *IEEE Trans. on ASSP*, 37:1641 – 1648, 1989.

[24] J.R.S. Makhoul and H. Gish. "Vector Quantization in Speech Coding". *Proceedings of the IEEE*, 73:1551 – 1588, 1985.

[25] C.S. Myers and L.R. Rabiner. "A level building dynamic time warping algorithm for connected word recognition". *IEEE Trans. on ASSP*, pages 284–297, 1988.

[26] D.B. Paul. "The Lincoln robust continuous speech recognizer". In *Proc. ICASSP, Glasgow, Scotland*, pages 449–452, 1989.

[27] L.R. Rabiner and B.H. Juang. *"Fundamentals of speech recognition"*. Prentice-Hall International Editions, 1993.

[28] S. Renals, N. Morgan, M. Cohen, and H. Franco. "Connectionist probability estimation in the Decipher speech recognition system". In *ICASSP*, pages 413 – 416, 1990.

[29] T. Robinson and F. Fallside. "A recurrent error propagation network speech recognition system". *Computer speech and language*, pages 259–274, 1991.

[30] A.E. Rosenberg, L.R. Rabiner, J.G. Wilpon, and D. Kahn. "Demisyllable based isolated word recognition". *IEEE Trans. on ASSP*, pages 137–165, 1994.

[31] H. Sakoe. "Two level DP matching - A Dynamic Programming based pattern matching algorithm for connected word recognition". *IEEE Trans. on ASSP*, pages 588–595, 1989.

[32] R. Schwartz. "The BBN BYBLOS continuous speech recognition system". In *Proc. DARPA Speech and Natural Language Workshop*, pages 94–99, 1989.

[33] K. Shikano, K. Lee, and D.R. Reddy. "Speaker Adaptation through vector quantization". In *Proc. ICASSP*, pages 234–238, 1989.

[34] D. Whitley and C. Bogart. "The evolution of connectivity: Pruning neural networks using genetic algorithms". In *International Joint Conference on Neural Networks*, volume 1, pages 134 – 137, 1993.

[35] B. Widrow. *"Adaptive Signal Processing"*. Prentice Hall : Englewood Cliffs, N.J., 1985.

[36] P.C. Woodland and S.G. Smyth. "An experimental comparison of connectionist and conventional classification systems on natural data". *Speech Communication*, pages 73 – 82, 1990.

[37] E.J. Yannakoudakis. "An assessment of N-phoneme statistics in phoneme guessing algorithms which aim to incorporate phontactic constraints". *Speech Communication*, (11), 1992.

[38] Y. Zhao. "An acoustic - phonetic based speaker adapation technique for improving speaker - independent continuous speech recognition". *IEEE Trans. on Speech and Audio processing*, 1994.

[39] V. Zue, J. Glass, M. Philips, and S. Seneff. "The summit speech recognition system". In *Proc. of DARPA Speech and natural language workshop*. Morgan Kaufmann, San Mateo, 1989.

[40] V. W. Zue. "The use of speech knowledge in automatic speech recognition". In *Readings in speech recognition.* Morgan Kaufmann Publishers, Inc., 1985.